Front. Comput. Sci., 2018, 12(2): 245–263
https://doi.org/10.1007/s11704-017-6325-0

RESEARCH ARTICLE

# Online clustering of streaming trajectories

**Jiali MAO (✉)[1,2], Qiuge SONG[1], Cheqing JIN[1], Zhigang ZHANG[1], Aoying ZHOU[1]**

1 School of Data Science and Engineering, East China Normal University, Shanghai 200062, China
2 School of Computing, China West Normal University, Nanchong 637009, China



**Abstract** With the increasing availability of modern mobile devices and location acquisition technologies, massive trajectory data of moving objects are collected continuously in a streaming manner. Clustering streaming trajectories facilitates finding the representative paths or common moving trends shared by different objects in real time. Although data stream clustering has been studied extensively in the past decade, little effort has been devoted to dealing with streaming trajectories. The main challenge lies in the strict space and time complexities of processing the continuously arriving trajectory data, combined with the difficulty of concept drift. To address this issue, we present two novel synopsis structures to extract the clustering characteristics of trajectories, and develop an incremental algorithm for the online clustering of streaming trajectories (called OCluST). It contains a *micro-clustering* component to cluster and summarize the most recent sets of trajectory line segments at each time instant, and a *macro-clustering* component to build large macro-clusters based on micro-clusters over a specified time horizon. Finally, we conduct extensive experiments on four real data sets to evaluate the effectiveness and efficiency of OCluST, and compare it with other congeneric algorithms. Experimental results show that OCluST can achieve superior performance in clustering streaming trajectories.

**Keywords** streaming trajectory, synopsis data structure, concept drift, sliding window

Received June 20, 2016; accepted May 19, 2017

E-mail: jlmao1231@stu.ecnu.edu.cn

## 1 Introduction

With the widespread application of modern mobile devices and the vigorous development of location acquisition technologies, numerous moving objects relay their locations continuously, and hence a tremendous amount of positional information is accumulated in a streaming manner. For instance, taxis equipped with GPS sensors transmit their locations to a data center at regular intervals so that the taxi company is capable of dispatching taxis efficiently. Trajectory data analysis enables us to discover the evolutionary moving behaviors of moving objects, which fosters a broad range of critical applications such as location-based social networks [1], route planning [2,3], intelligent transportation management [4], and road infrastructure optimization [5].

As a typical class of the moving pattern discovery approach, clustering aims to group a large number of trajectories into comparatively homogeneous clusters to extract the representative paths or common moving trends shared by various objects [6–12]. For instance, in hurricane landfall forecast applications, discovering the common behavior of hurricanes can improve the forecasting accuracy. In animal migration analyzing applications, extracting the common behaviors of animals can reveal the cause of animal migration. An important observation is that the trajectory of one moving object may belong to different clusters with the progression of a stream. That is, the clustering result may evolve with time. For example, traffic is highly dynamic in a road network scenario; accordingly, the trajectory stream generated by vehicles may evolve more dramatically over shorter time horizons. Figure 1 illustrates a small example of four taxis' tra-

jectories from time instant $t_0$ to $t_2$. Obviously, during $[t_0, t_1]$, four trajectories construct one cluster, while in $[t_1, t_2]$, they form two clusters owing to different moving directions.

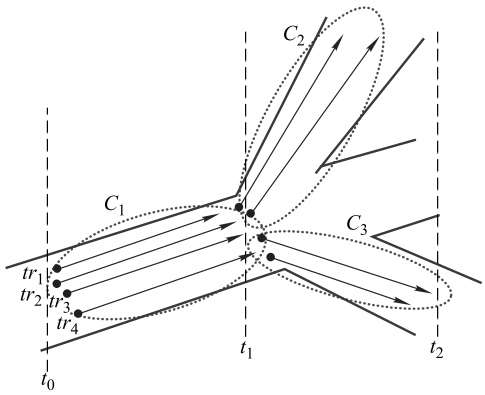


**Fig. 1** Evolution of clusters

To estimate real-time traffic conditions of a road, we need to keep up with the continuously evolving streaming trajectories. This requires that clustering is performed online on a recent subset of the streaming data. The sliding-window model that keeps only the most recent $W$ tuples in a limited time window is suitable for describing the evolution of the clustering results. In other words, any tuple outside of the current time window has no effect on the current clustering results. On the basis of the sliding-window model, we can focus on the traces of vehicles in recent time horizons, and generate clusters based on them. By extracting the moving behavior of each cluster at each time instant, we can estimate real-time traffic conditions of the road where each cluster is located. Such traffic information can be provided to road users and applied to real-time routing planning and traffic management.

To the best of our knowledge, the problem of online clustering of trajectory data streams using the sliding-window model has not been addressed to date. The challenges come from limited system resources, huge stream volumes, high arrival rates, and real-time response requirements, combined with the difficulty of concept drift. Furthermore, the existing work cannot be adopted to deal with this issue directly. First, the existing work on trajectory data stream clustering does not consider removing the influence of outdated tuples [12]. Along with incrementally clustering incoming trajectory data, the expired records cannot be discarded upon the arbitrary arrival of new ones. The size of micro-clusters continuously increases with the drifts of the cluster centers, which leads to concept drift and thus degrades the clustering performance. Second, the existing data stream clustering using the sliding-window model treats each tuple as a *full* entry [13], while each tuple in the trajectory data stream is only a *part* of an entry.

In this paper, we develop an online approach to deal with this issue, including the line segment *micro-clustering* component and *macro-clustering* component. During the micro-clustering phase, a number of micro-clusters, each represented by a compact synopsis data structure, are maintained incrementally. During the macro-clustering phase, a small number of macro-clusters are built upon the micro-clusters according to a clustering request within a specified time horizon. The core innovation in this framework is exploiting a novel compact synopsis data structure to represent each micro-cluster. Distinct from trajectory simplification [6] or trajectory compression [14] technologies, such a synopsis can preserve the spatio-temporal characteristics of the trajectory in memory while removing the influence of obsolete tuples. The experimental results demonstrate the applicability of the new proposal for online clustering streaming trajectories, and show better clustering performance than the congeneric algorithms. The contributions of this paper are summarized below.

- We address the problem of online clustering streaming trajectories using the sliding-window model. Neither existing trajectory clustering algorithms nor data stream clustering algorithms using the sliding-window model can be adopted to tackle this issue directly.
- We exploit two new synopsis data structures to extract and maintain the spatio-temporal clustering characteristics of sets of trajectories incrementally.
- We propose an online algorithm, called OCluST, to cluster streaming trajectories over the sliding-window model, which incrementally summarizes clusters, and safely eliminates the expired tuples.
- We conduct a comprehensive series of experiments on four real data sets to manifest the efficiency and effectiveness of our proposal, as well as its superiority over other congeneric approaches.

The rest of this paper is organized as follows. In Section 2, the problem is defined formally. In Section 3, we outline and analytically study the scheme for clustering streaming trajectories, called OCluST. In Section 4, a series of comparison experiments are conducted on real data sets to evaluate OCluST through comparisons with other congeneric algorithms. In Section 5, we review the latest work related to OCluST. Finally, in the last section, we conclude this article in brief and discuss the direction of future study.

## 2 Problem definition

In this section, we define some notations formally, as listed in Table 1, and then formalize the problem of online clustering of streaming trajectories.

**Table 1** List of notations

| Notation | Explanation |
|---|---|
| $S$ | Trajectory stream |
| $W$ | Window size |
| $p_i^{(j)}$ | Location of an object $o^{(j)}$ at time instant $t_i$ |
| $L_i$ | Line segment connecting two adjacent points |
| $n$ | Number of line segments |
| $m$ | Number of *TF*s in an *EF* |
| $C_i$ | Set of line segments at time instant $t_i$ |
| $cen$ | Central point |
| $\epsilon$ | Error threshold |
| $\theta$ | Angle of line segment |
| $k$ | Maximum number of *EF*s |
| $\rho$ | Time tolerance threshold |
| $\gamma$ | Distance threshold |
| $\lambda$ | Spatial proximity weight |

**Definition 1** (Trajectory stream) A trajectory stream that consists of a series of positional records of $M$ moving objects is denoted as $S = \{(o^{(1)}, p_1^{(1)}), (o^{(2)}, p_1^{(2)}), \ldots, (o^{(M)}, p_1^{(M)}), (o^{(1)}, p_2^{(1)}), \ldots\}$, where $p_i^{(j)}$ is the location of an object $o^{(j)}$ at time instant $t_i$ in 2D space (i.e, $p_i^{(j)} = (x_i^{(j)}, y_i^{(j)})$).

A trajectory stream is commonly assumed to be unbounded. Hence, it is common to use a window to limit an infinite stream to a specified finite set of records within a given time horizon. We adopt the idea of a sliding tuple-based window model, which is commonly used for discounting obsolete data, and only the latest $W$ records in the window are valid at the point in time of the clustering.

**Definition 2** (Sliding tuple-based window) Given a stream $S$, a window size $W$, a starting time instant $t_s$, and an ending time instant $t_e$, a sliding tuple-based window over $S$ at $t_e$ is a finite multiset of stream elements with $S_W = \{(o^{(1)}, p_s^{(1)}), (o^{(2)}, p_s^{(2)}), \ldots, (o^{(M)}, p_s^{(M)}), \ldots, (o^{(1)}, p_e^{(1)}), \ldots, (o^{(M)}, p_e^{(M)})\}$. Whenever the window slides forward, the number of positional elements received in $S_W$ is $W$.

In general, the positional record (represented as a trajectory) of one object is defined as follows.

**Definition 3** (Trajectory) The trajectory of an object $o$, denoted as $tr_o = \{(p_1, t_1), (p_2, t_2), \ldots\}$, is a subsequence of $S$ affiliated with $o$. Such records arrive in chronological order, i.e, $\forall i < j$, $t_i < t_j$. Two temporal adjacent points are connected into a line segment $L_i$, i.e., $L_i$ is denoted as $(p_i, p_{i+1})$. Correspondingly, the trajectory of an object $o$ is also denoted as $tr_o = \{(L_1, t_1), (L_2, t_2), \ldots\}$.

Owing to the infeasibility of keeping massive trajectory data in memory, it is necessary to summarize original data and approximate the clustering results. We summarize original data in memory using a compact synopsis data structure called the Exponential Histogram of Temporal Trajectory Cluster Feature (*EF*). Each bucket in an *EF* is a Temporal Trajectory Cluster Feature (*TF*), which attempts to summarize the features of incoming trajectory line segments at each time instant. Accordingly, we use a Minimum Bounding Rectangle (MBR) to represent the spatial range of all line segments that are contained in a *TF*.

**Definition 4** (Temporal trajectory cluster feature (*TF*)) *TF* of a set of line segments $C = \{L_1, L_2, \ldots, L_n\}$, is of the form $(LS_{cen}, LS_A, LS_{len}, SS_{len}, \max_{len}, \min_{len}, BL, TR, n, t)$.

- $LS_{cen}$: linear sum of the line segments' center points;
- $LS_A$: linear sum of the product of the line segments' angle and length;
- $LS_{len}$: linear sum of the line segments' length;
- $SS_{len}$: squared sum of the line segments' length;
- $\max_{len}$: maximum of the line segments' length;
- $\min_{len}$: minimum of the line segments' length;
- $BL$: bottom left corner of MBR;
- $TR$: top right corner of MBR;
- $n$: number of line segments;
- $t$: time stamp of the last line segment;

Figure 2(a) illustrates an example of MBR that contains three black lines. We use a line (denoted as $TF.rp(s, e)$, where $s$ and $e$ are the starting and the ending points, respectively) to represent the moving pattern of all line segments in *TF* in terms of the central point (denoted as $TF.cen$) and the angle (denoted as $TF.\theta$). Specifically, for angle calculation, as a longer line segment is intrinsically more important than a shorter line segment, we take the line segment's length into account in obtaining the angle of the representative line segment, i.e., $LS_A/LS_{len}$. Additionally, it is easy to derive a shorter representative trajectory line segment owing to the effects of a few short line segments that are absorbed in a cluster. Hence, we attempt to extend the average length of line segments to make a representative line segment conform

more accurately to the original data distribution. We extend the average length of the line segments with the product of the standard deviation (denoted as $\sigma$) and the ratio of $\max_{len}$ to $\min_{len}$. Then, we derive $TF.rp(s, e)$ by the following equations:

$$s = ((cen)^{(1)} - \frac{len}{2}\cos\theta, (cen)^{(2)} - \frac{len}{2}\sin\theta),$$

$$e = ((cen)^{(1)} + \frac{len}{2}\cos\theta, (cen)^{(2)} + \frac{len}{2}\sin\theta),$$

where

$$cen = \frac{LS_{cen}}{n}, \theta = \frac{LS_A}{LS_{len}}, len = \frac{LS_{len}}{n} + \frac{\max_{len}}{\min_{len}} \cdot \sigma,$$

$$\sigma = \frac{\sqrt{n \cdot SS_{len} - LS_{len}^2}}{n}.$$

As illustrated in Fig. 2(a), the three line segments $(tr_1, tr_2, and tr_3)$ have different lengths and angles. We compute the central point, the angle, and the average length of the three line segments to obtain a representative line segment, namely, a blue thick line with red dashed part (denoted as $(TF.rp_s, TF.rp_e)$).

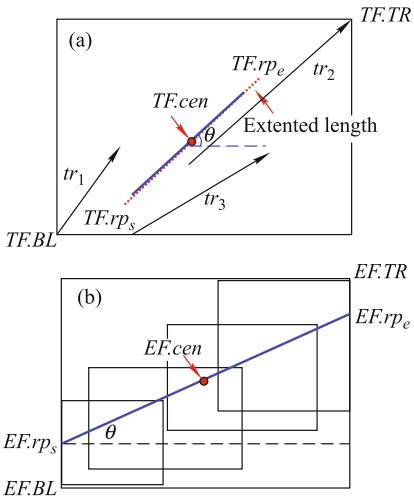


**Fig. 2** Representative line segment. (a) $TF.rp$; (b) $EF.rp$.

**Property 1** (Additive property) Let $TF(C_1)$ and $TF(C_2)$ denote the $TF$ structures for two sets $C_1$ and $C_2$, respectively, and $C_1 \cap C_2 = \emptyset$. $TF(C_1 \cup C_2)$ is constructed on $TF(C_1)$ and $TF(C_2)$. The new entries $LS_{cen}$, $LS_A$, $LS_{len}$, $SS_{len}$, and $n$ are equal to the sum of the corresponding entries in $TF(C_1)$ and $TF(C_2)$. The new entries $\max_{len}$ and $t$ are computed as the maximum of the corresponding entries in $TF(C_1)$ and $TF(C_2)$. The new entry $\min_{len}$ is computed as $\min(TF(C_1).\min_{len}, TF(C_2).\min_{len})$. Moreover, the corners of the new $TF$ can be directly computed based on the two original corners.

As a $TF$ may consist of multiple line segments, and they will go out of the window one by one in the future, this necessitates a structure to deal with the expired line segments. An Exponential Histogram (EH) is a well-known approach to deal with the sliding-window model, where all data in the stream are split into many buckets in terms of the arrival time [15]. Inspired by the EH, we exploit a synopsis structure called $EF$ below.

**Definition 5** (Exponential histogram of temporal trajectory cluster Feature ($EF$)) Given an error threshold $\epsilon$, $EF$ is a collection of multilevel $TF$s on the sets of trajectory line segments $C_1, C_2, \ldots$ with the following constraints:

1) $\forall i < j$, any trajectory 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 $\lceil \frac{1}{\epsilon} \rceil + 1$ $TF$s are placed in each level.

**Lemma 1** Given an $EF$ that contains $n_i$ tuples, and an error threshold $\epsilon$, the number of obsolete tuples is within $[0, \epsilon n_i]$, and the number of $TF$s is at most $(\frac{1}{\epsilon} + 1)(\log(\epsilon n_i + 1) + 1)$.

**Proof** Only the last $TF$ (namely, the oldest $TF$ at the highest level of $TF$s) in $EF$ may contain the expired records. Let $n_s$ denote the number of tuples in the last $TF$. According to Definition 5, we have $\frac{1}{\epsilon}(1 + 2 + 4 + \cdots + n_s) \leqslant n_i$. Then, $n_s \leqslant \epsilon n_i$ holds. Moreover, an EH structure with a window size $n_i$ and parameter $\epsilon$ can be constructed. Each $TF$ maps to a bucket in EH structure. According to [15], EH structure computes an $\epsilon$-deficient synopsis using at most $(\frac{1}{\epsilon} + 1)(\log(\epsilon W + 1) + 1)$ buckets, where $W$ represents the window size. Thus, there exist at most $(\frac{1}{\epsilon} + 1)(\log(\epsilon n_i + 1) + 1)$ $TF$s in an $EF$.

We maintain $EF$ in the following way. When a new line segment is incorporated into an existing $EF$, a new 0-level $TF$ will be generated for it at first. After absorbing more and more line segments, once the number of 0-level $TF$s in $EF$ exceeds the threshold $(\lceil \frac{1}{\epsilon} \rceil + 1)$, the two oldest 0-level $TF$s are merged to generate a 1-level $TF$. Such a merge operation may propagate from the lowest level toward the higher levels until, at a certain level, there are fewer than $\lceil \frac{1}{\epsilon} \rceil + 1$ $TF$s.

Figure 3 explains how to maintain an $EF$ when $\epsilon = 1/3$. This means that at most four $TF$s are kept at each level. As we can see, when $L_5$ arrives, a new 0-level $TF$ $(TF(\{L_5\}))$ is generated, which results in five 0-level $TF$s. Then, a 1-level $TF$ $(TF(\{L_1, L_2\}))$ is generated by merging $TF(\{L_1\})$ and $TF(\{L_2\})$. A similar merging operation occurs when $L_7$ arrives. Furthermore, the arrival of $L_{13}$ triggers the merging of

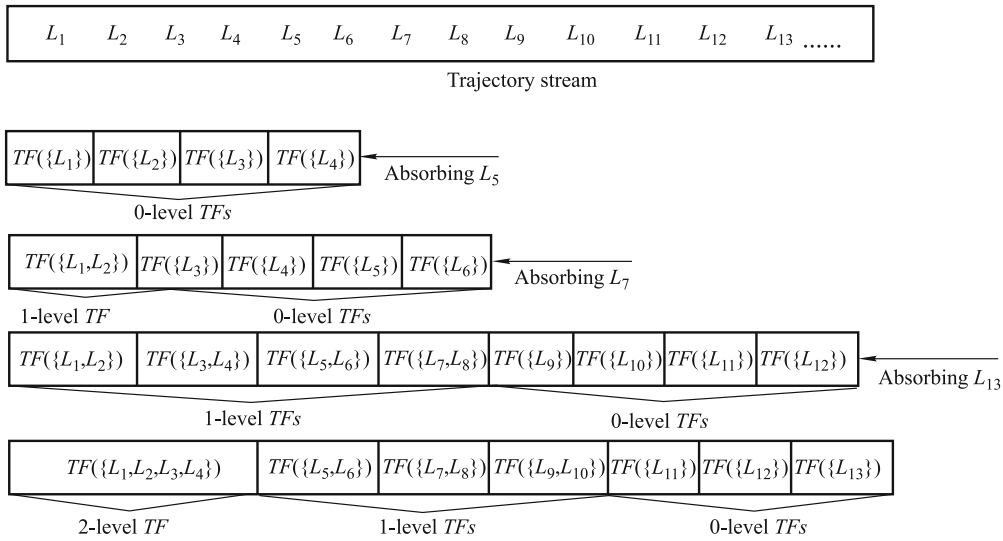


**Fig. 3** Process of incorporating line segments into an *EF* ($\epsilon = 1/3$)

$TF(\{L_9\})$ and $TF(\{L_{10}\})$, which further triggers the merging of $TF(\{L_1, L_2\})$ and $TF(\{L_3, L_4\})$.

Likewise, to extract the moving pattern of all line segments absorbed in *EF*, we derive a representative line segment for each *EF*. The central point (denoted as *EF.cen*) and the angle (denoted as *EF.θ*) of the representative line segment are computed by the following equations, respectively:

$$EF.cen = \frac{\sum_i TF_i.LS_{cen} \times TF_i.n}{\sum_i TF_i.n}, \quad EF.\theta = \frac{\sum_i TF_i.LS_A}{\sum_i TF_i.LS_{len}}.$$

The corners of MBR that describe the *EF* are computed based on all *TF*s' MBRs. Then, we plot a line across *EF.cen*, along *EF.θ*, and finally extend it to the borders of MBR. The intersection points are treated as the starting and ending points of the representative line segment. Figure 2(b) illustrates an example of the MBRs for a group of *TF*s contained in an *EF*, and the blue thick line segment ($EF.rp_s$, $EF.rp_e$) is the derived representative line segment.

The *TF* and *EF* synopsis structures allow us to effectively extract and maintain the spatio-temporal characteristics of clusters at different intervals. Additionally, the *EF* synopsis can promptly remove the obsolete records when clustering trajectories, which prevents the size of each cluster from getting larger and larger, and hence avoids the shifting of the cluster's center. Figure 4 illustrates the evolutions of three clusters (represented by $EF_1$, $EF_2$, and $EF_3$) in the current window (including seven time instants). Owing to the elimination of obsolete data as time goes by, the boundary of each cluster does not become larger and larger, and hence avoids the concept shift. Meanwhile, since each *EF* maintains its histogram, the *EF* synopsis can adjust the frequency of generating a new *TF* adaptively, which enables a reduction in space consumption. Specifically, a new *EF* that contains a *TF* will be created only when the incoming trajectory segment cannot be absorbed into any existing cluster. If such a segment is an outlier, only one *TF* will be created for it in a new *EF*, and the *EF* that contains the outlier will be removed when it becomes outdated.

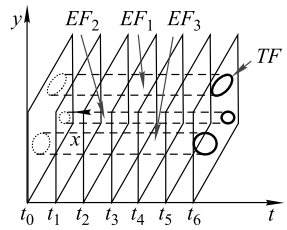


**Fig. 4** Example of *EF* in current window

Our goal is to divide all trajectory line segments into clusters in terms of the similarity measurement. Whether the similarity measurement between a trajectory line segment and an *EF*, or that among *EF*s (an *EF* is represented by its representative line segment), is essentially translated to measure the similarity between trajectory line segments. Note that two trajectory line segments close to each other but generated at different time intervals are actually not similar. In view of this, we define similarity as the combination of the spatial proximity and temporal closeness between line segments. For a spatial proximity measurement, although Euclidean-distance-based schemes are commonly used in the spatial data management field, including Dynamic Time Warping Distance (DTW) [16], Longest Common Subsequences (LCSS) [17], Edit Distance with Real Penalty (ERP) [18], and Edit Dis-

tance in Real Sequence (EDR) [19], they are inappropriate to measure the spatial proximity in some scenarios with bidirectional properties. For instance, as illustrated in Fig. 5, there are three trajectories ($tr_1$, $tr_2$, and $tr_3$) on two separate roads. According to Euclidean-distance-based schemes, $tr_1$ is closer to $tr_2$. Nevertheless, in a real road network, there exists no route between $tr_1$ and $tr_2$, and accordingly, $tr_2$ is closer to $tr_3$. Hence, we leverage the adapted Hausdorff distance in [20] to measure the spatial proximity. The distance between two trajectory line segments is regarded as 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, $p'_s$ (or $p''_s$) and $p'_e$ (or $p''_e$) denote the starting and ending positions of two line segments separately, and $\|p'_s, p''_s\|$ denotes the length of the shortest path between $p'_s$ and $p''_s$.

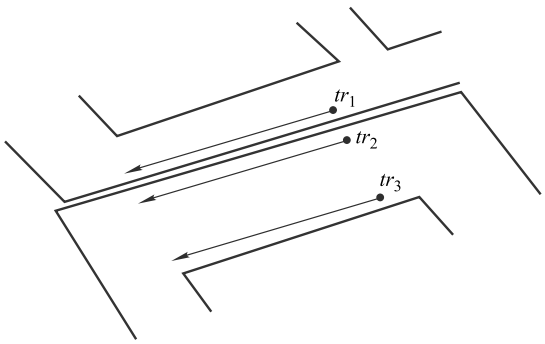


**Fig. 5** Spacial proximity in road network scenario

On the basis of a spatial proximity measurement, we define the difference between two trajectory line segments as follows.

**Definition 6** (Difference measurement) Given a temporal closeness threshold $\rho$ ($0 < \rho \leqslant 1/2$), a spatial proximity threshold $\gamma$ ($\gamma \leqslant 1$), a spatial proximity weight $\lambda$ ($0 \leqslant \lambda \leqslant 1$), a window size $W$, and the arrival time instant of line segment $t$, the lengths of two line segments (denoted as $L'$ and $L''$) are denoted as $\|L'\|$ and $\|L''\|$ separately. If $DL(L', L'')/(\|L'\| + \|L''\|) \leqslant \gamma$ and $(L'.t - L''.t < \rho W)$, the difference between $L'$ and $L''$ is defined as follows:

$$Diff(L', L'') = \lambda \cdot \frac{DL(L', L'')}{\|L'\| + \|L''\|} + (1 - \lambda) \cdot (L'.t - L''.t).$$

We usually put more weight on spatial proximity by setting $\lambda > 1/2$. Generally, a smaller $Diff$ value means a high similarity between line segments, namely, the line segments are closer to each other in recent time intervals.

Finally, we summarize the problem definition below:

Given a time horizon of length *len*, the current time instant $t_c$, and a trajectory stream portion that flows into the current time window (depicted as a multiset of trajectory segments), our goal is to discover the micro-clusters (represented as a set of *EF*s) at each time instant, and the macro-clusters according to a clustering request within the specific time interval $[t_c - len, t_c]$.

In the following section, on the basis of the *TF* and *EF* synopsis, we study the problem of online clustering streaming trajectories using the sliding-window model.

## 3 OCluST algorithm: online clustering of streaming trajectories

In this section, we propose an online approach to cluster streaming trajectories using the sliding-window model, called Online Clustering of Streaming Trajectories (OCluST). Since a trajectory is referred to as a set of consecutive line segments (Definition 3), our scheme is essentially a line segment clustering algorithm. As illustrated in Algorithm 1, OCluST is comprised of two components: a line segment micro-clustering phase (line 1) and an *EF*s macro-clustering phase (line 2). During the first phase, on the basis of the aforementioned data structures (*TF* and *EF*), appropriate statistical information of the micro-clusters in the current time window is extracted and maintained incrementally, as shown in Algorithm 2. Each micro-cluster is represented by an *EF* structure (Definition 5), and each bucket in an *EF* is a *TF* (Definition 4) that represents the summary statistics of a set of trajectory segments at each time instant. During the second phase, given a time horizon, a small number of macro-clusters are derived on *EF*s by invoking traditional density-based clustering techniques. In this way, the cluster characteristics of streaming trajectories can be preserved with continuity in time and contiguity in space.

### 3.1 Line segment micro-clustering

The micro-clustering phase aims to cluster the continuously arriving trajectory line segments at each time instant while discarding expired ones. Algorithm 2 shows the main framework to generate and maintain *EF*s for trajectory line segments. It executes two algorithms to achieve the subtasks separately (lines 7 and 17). Let $Z$ represent the set of *EF*s generated during the micro-clustering phase. Initially, $Z$ is emptied, and subsequently, $k$ *EF*s are generated one after another and added to $Z$ when continuously receiving $k$ line segments.

Specifically, we create an *EF* that contains a *TF* for each incoming line segment through an initialization process, and regard the line segment itself as the representative line segment of such an *EF* (line 2). Instead of the orginal trajectory data is kept in memory, at most $k$ *EF*s are kept in memory at any time.

**Algorithm 1** OCluST

**Input**: $S$: stream of trajectory line segments; $W$: window size; $\epsilon$: error threshold; $\rho$: time tolerance threshold; $k$: maximum number of EFs; *len*: time horizon of length; $t_c$: current time intant;

**Output**: $Z'$: set of all generated *EF*s;

1 $Z \leftarrow MicroClu(S, W, \epsilon, \rho, k)$;
2 Execute *macro-clustering on Z within* $[t_c - len, t_c]$;
3 Get a set $Z'$ using the macro-clustering result;
4 **return** $Z'$;

**Algorithm 2** $MicroClu(S, W, \epsilon, \rho, k)$

**Input**: $S$: trajectory stream; $W$: window size; $\epsilon$: error threshold; $\rho$: time tolerance threshold; $k$: maximum number of *EF*s;

**Output**: $Z$: set of all generated *EF*s;

1 $Z \leftarrow \emptyset$; $Z_{list} \leftarrow \emptyset$;
2 Initialize $Z$;
3 **foreach** line segment $L_x$ in $S$ **do**
4 find the most similar *EF*, $h$, and the second most similar *EF*, $h_s$, for $L_x$;
5 **if** $Diff(h.rp, L_x) < d_{\min}$ **then**
6 /*$d_{\min}$ denotes the minimum difference threshold of absorbing trajectory segments into an *EF**/;
7 $Insertline(L_x, h, h_s, \epsilon)$;
8 **else**
9 **if** $|Z| = k$ **then**
10 **if** $\exists h_e \in Z$, $((|t_c - h_e.t| \geqslant W) \vee ((|t_c - h_e.t| \geqslant \rho W) \wedge (h_e.n_i \leqslant \frac{\sum_{i=1}^{k} n_i}{k})))$ **then**
11 /* $n_i$ denotes the number of an *EF* */;
12 $Z \leftarrow Z \setminus \{h_e\}$;
13 /*$Z_{list}$ denotes a set of influenced *EF*s owing to the deleting, merging, and creation of some *EF*s; */;
14 $Z_{list} \leftarrow Z_{list} \cup$ the *EF*s where the most similar *EF* is $h_e$;
15 **else**
16 find the most similar *EF* pair $(h_i, h_j)$;
17 $MergeEF(h_i, h_j, \epsilon, Z)$;
18 $Z_{list} \leftarrow Z_{list} \cup$ the *EF*s where the most similar *EF* is $h_i$ or $h_j$;
19 $h_n \leftarrow newEF(TF(\{L_x\}))$;
20 $Z \leftarrow Z \cup h_n$;
21 $Z_{list} \leftarrow Z_{list} \cup h_n$;
22 $Maintain(Z_{list})$;
23 **return** $Z$;

When a line segment $L_x$ arrives, we attempt to find its most similar *EF* from the existing *EF*s. According to Definition 3, only *EF* (denoted as $h$) with the greatest spatial proximity over recent time interval is deemed as the appropriate *EF* to absorb $L_x$, and the entries of it are accordingly adjusted based on $L_x$. If $L_x$ cannot find its most similar *EF*, a new *EF* $h_n$ that only contains $TF(\{L_x\})$ will be created (line 19) on the condition that the number of *EF*s is less than $k$. When the number of *EF*s exceeds $k$, we need to take into account eliminating the expired *EF*s (lines 10–12) or merging *EF*s (lines 16 and 17) to make room for the newly created EF. The detailed procedure for inserting $L_x$ into $h$ is shown in Algorithm 3. A $TF_0(\{L_x\})$ is generated first and is added to $h$ (line 1). Subsequently, once the number of 0-level *TF*s in $h$ exceeds $\lceil \frac{1}{\epsilon} \rceil + 1$, the two oldest 0-level *TF*s are merged to generate a 1-level *TF*. Such a merge operation is repeated several times for higher levels until the number of a certain level of *TF*s lower than $\lceil \frac{1}{\epsilon} \rceil + 1$ (lines 4–10).

**Algorithm 3** $Insertline(L_x, h, h_s, \epsilon)$

**Input**: $L_x$: newly arrived line segment; $h$: most similar *EF* of $L_x$; $h_s$: second most similar *EF* of $L_x$; $\epsilon$: error threshold;

**Output**: $h$;

1 $h \leftarrow h \cup \{TF_0(\{L_x\})\}$;
2 $h.t \leftarrow L_x.t$;
3 $l \leftarrow 0$;
4 **while** *true* **do**
5 **if** $n_l \leqslant \lceil \frac{1}{\epsilon} \rceil + 1$ **then**
6 /* $n_l$ denotes the number of $l$-level *TF*s */;
7 break;
8 **else**
9 Merge two oldest $l$-level *TF*s of $h$ into an $(l+1)$-level *TF*;
10 $l \leftarrow l + 1$;
11 **if** the oldest *TF* of $h$,$TF_{lst}$,expires **then**
12 Drop $TF_{lst}$ from $h$;
13 **if** $(Diff(h.rp, h_s.rp) < h.ds)$ **then**
14 $h.ds \leftarrow Diff(h.rp, h_s.rp)$;
15 $h.c \leftarrow h_s$;
16 **return** $h$;

In the following, we describe the process of individual subroutines such as eliminating obsolete records and merging *EF*s.

### 3.1.1 Expired records elimination

In essence, the significance of each position in a trajectory stream is time-decaying, until it finally becomes outdated and negligible. To eliminate the adverse effect of expired records on the micro-clustering results (e.g., concept drift), we consider removing the insignificant records including obsolete *EF*s, and a few *EF*s with the earlier updated time which will no longer absorb tuples in the current window. More specifically, when a line segment $L_x$ is incorporated into its most similar *EF* $h$, $h$ must be checked to see if it contains obsolete *TF*s, and then discard them (lines 11 and 12 in Algorithm 3). Further, when the number of *EF*s exceeds the specified

threshold $k$, we not only remove the expired *EF*s but also filter out *EF*s with the earlier updated time and an insufficient amount of participating trajectory segments. Let $t_i$ represent the time instant of the newest updated *TF* in an *EF* $h_i$, and $t_c$ represent the current time instant. Then, we can obtain the updated period of $h_i$ by $(t_c - t_i + 1)$. If the updated period is beyond more than half (e.g., 2/3) of the current window size, $h_i$ is deemed as an *EF* that has not been updated for a long time. Further, we filter out such *EF*s with longer updated periods and participating records smaller than the average participating records of all *EF*s in the current window (lines 10 and 12 in Algorithm 2).

### 3.1.2 *EF* merging

If no *EF* is deleted, that is, none of the aforementioned eliminating criteria are met, we attempt to find the most similar *EF* pair to merge until the size of the existing *EF*s meets the space constraints, as shown in Algorithm 4. Only the most similar *EF* pair will be merged into a new *EF* according to Definition 6. However, even the two most similar *EF*s face the problem of an inconsistent time instant of corresponding level *TF*s. Therefore, for any two most similar *EF*s $h_i$ and $h_j$, we first align all level *TF*s of them in terms of the time range of each level of *TF*s (line 1). To be specific, we obtain a set of time instants of all levels of *TF*s in $h_i$ and $h_j$, and insert $h_i$ or $h_j$ with the empty level of the missing time range to ensure one-to-one correspondence between all levels of *TF*s of $h_i$ and $h_j$. Later, the merging process is akin to the process of incorporating the line segments into an *EF*. If the number of corresponding $l-$*level TF*s in two *EF*s exceeds $\lceil \frac{1}{\epsilon} \rceil + 1$, the oldest $l-$*level TF*s are merged into $(l+1)-$*level TF*s (lines 5 and 6). Otherwise, $l-$*level TF*s in two *EF*s are directly combined into $l-$*level TF*s of new *EF* (lines 7 and 8). Such operations will cascade to a higher level $l = 0, 1, 2, \ldots$, until the sum of the number of a certain level of *TF*s is 0.

However, the computation overhead of finding the most similar *EF* pair is costly, especially when the number of *EF*s ($k$) is too large. A nested loop used for calculating and comparing the similarity between all pairs of *EF*s is inevitable, and costs $O(k^2)$ ($k$ is the number of *EF*s). Owing to the evolving properties and high updating cost of a data stream, tree-based indexes cannot be well applied to cluster trajectory streams. We opt for a heuristic strategy to speed up this process. For each *EF*, we maintain its most similar *EF* (denoted as $c$) as well as the difference (denoted as $ds$) between its most similar *EF* and itself. In particular, when receiving a line segment $L_x$, we attempt to search its most similar *EF* $h$ and the second most similar *EF* $h_s$ (lines 3 and 4 in Algorithm 2). During the process of absorbing $L_x$ into $h$, the difference between $h$ and $h_s$ (denoted as $Diff(h.rp, h_s.rp)$) is computed and compared with $h.ds$. If $Diff(h.rp, h_s.rp) \leqslant h.ds$, the original most similar *EF* of $h$ ($h.c$) is replaced with $h_s$ (lines 13–15 in Algorithm 3). Only when a new *EF* is created for $L_x$, and the most similar *EF* of $h$ is eliminated or merged into the other *EF*, we need to search the most similar *EF* for the influenced *EF*s (denoted as $Z_{list}$) by the difference measurement between the representative trajectory line segments of the *EF*s. Hence, we derive $Z_{list}$ after newly creating, deleting, and merging the *EF*s (lines 14, 18, and 21 in Algorithm 2), and implement the maintenance of the most similar *EF*s for $Z_{list}$ (line 22 in Algorithm 2). The maintaining details are illustrated in Algorithm 5. For each *EF* $h_i$ in $Z_{list}$, we re-find the most similar *EF* for it by comparing the difference between $h_i$ and every existing *EF* $h_j$ (lines 1–5). At the same time, we accordingly update $h_j$'s most similar *EF* via a difference comparing process (lines 6–8). In this way, when searching for the most similar *EF* pair to merge, we simply need to traverse the most similar *EF* lists of all *EF*s to find the *EF* pair with the least difference. As a result, the cost of searching the most similar *EF* pair to merge in the best case can be reduced to $O(k)$. Only when the deleted or merged *EF* is the most similar *EF* of all *EF*s, the cost of searching the most similar *EF* pair to merge is $O(k^2)$. Actually, this extreme case rarely occurs. Moreover, keeping the most similar *EF* for each *EF* does not require much extra space, which additionally needs to store the index of the most similar *EF*, the value of $ds$, and a list of indexes of the inverse most similar *EF*s for each *EF*. As compared to the memory consumption of *TF*s that are included in each *EF*, such extra space consumption is negligible if $n \gg k$.

**Algorithm 4** $MergeEF(h_i, h_j, \epsilon, Z)$

**Input**: $h_i, h_j$: two *EF*s that need to be merged; $\epsilon$: error threshold; $Z$: a set of all generated *EF*s;
**Output**: $Z$;
1 Align all level *TF*s of $h_i$ and $h_j$ in terms of their time range;
2 $l \leftarrow 0$;
3 **while** $(h_i.n_l + h_j.n_l + h_{new}.n_l > 0)$ **do**
4 /* $n_l$ denotes the number of $l$-level *TF*s */;
5 **if** $(h_i.n_l + h_j.n_l + h_{new}.n_l > \lceil \frac{1}{\epsilon} \rceil + 1)$ **then**
6 Merge the oldest $l$-level *TF*s from $h_i$ and $h_j$ into new $(l+1)$-level *TF*s of $h_{new}$;
7 **else**
8 Insert into $h_{new}$ with $l$-level *TF*s of $h_i$ and $h_j$;
9 $l \leftarrow l + 1$;
10 $Z \leftarrow Z \setminus \{h_i, h_j\} \cup \{h_{new}\}$;
11 **return** $Z$;

**Algorithm 5** *Maintain*($Z_{list}$)

**Input**: $Z_{list}$: set of influenced *EF*s;
**Output**: $Z_{list}.c$: most similar EFs for $Z_{list}$;
1 **foreach** $EF, h_i$ in $Z_{list}$ **do**
2  **foreach** $EF, h_j$ in $Z$ **do**
3  **if** $(Diff(h_i.rp, h_j.rp) < h_i.ds)$ **then**
4  $h_i.ds \leftarrow Diff(h_i.rp, h_j.rp)$;
5  $h_i.c \leftarrow h_j$;
6  **if** $(Diff(h_i.rp, h_j.rp) < h_j.ds)$ **then**
7  $h_j.ds \leftarrow Diff(h_i.rp, h_j.rp)$;
8  $h_j.c \leftarrow h_i$;
9 **return** $Z_{list}.c$;

### 3.2 *EF*s macro-clustering

Given a clustering request over a time horizon of length *len* and a current time instant $t_c$, we can further derive larger macro-clusters based on previously generated micro-clusters. As micro-clusters are line segment clusters of arbitrary shape, and density-based clustering methods are suitable for discovering such types of clusters, we implement macro-clustering using the DBSCAN method. As mentioned in Section 2, the micro-clusters are represented by the representative trajectory line segments of the *EF*s. Hence, *EF*s in $[t_c - len, t_c]$ are treated as pseudo line segments, and they are re-clustered to produce macro-clusters using a variant of the DBSCAN algorithm [6,12]. The approach of generating a representative trajectory for a macro-cluster is the same as that for a set of trajectory partitions in [6]. Specifically, a representative trajectory of a macro-cluster is a small sequence of points generated by using a sweep line approach and a smoothing technique, which can better capture the spatial characteristics of a macro-cluster. If the given time horizon exceeds the current window size, in addition to the latest *EF*s that are maintained in main memory, the historical *EF*s stored on disk within the specific period will also be used for macro-clustering. At this point, the representative trajectory line segments of such *EF*s can be clustered offline to generate the macro-clusters by executing the DBSCAN algorithm. Intuitively, larger macro-clusters tend to indicate more valuable patterns in contiguous regions during continuous periods. For instance, in the scenario of hot region identification, the road regions that the macro-clusters cover exhibit similar mobile behaviors (e.g., maintain a similar low speed) at consecutive intervals, which can be further identified as hot areas (e.g., traffic congestion).

### 3.3 Time and space complexity

The goal of OCluST is to cluster continuously arriving trajectory line segments. The cost of incorporating a new line segment $L_x$ into its most similar *EF* mainly involves lines 4, 9, and 16 in Algorithm 2. The cost of line 4 (finding the most similar *EF* for $L_x$) is simply $O(k)$. At line 9, when the number of *EF* exceeds $k$, the cost of removing obsolete *EF*s is $O(k)$. At line 16, the cost of calculating the difference between *EF*s and merging a similar pair of *EF*s is $O(k^2)$ for the worst-case scenario. Consequently, the per-record processing cost is at most $O(k^2)$. However, in essence, the merging process seldom occurs because the eliminating criteria ensure that some *EF*s are deleted to make enough room for newly created *EF*.

Concerning space complexity, given an error threshold $\epsilon$, maximal number of *EF* $k$, window size $W$, and 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 $(\frac{1}{\epsilon} + 1)(\log(\epsilon n_i + 1) + 1)$ *TF*s in an *EF* [15]. The total number of *TF*s in $k$ *EF*s is at most $\sum_{i=1}^{k}(\frac{1}{\epsilon} + 1)(\log(\epsilon n_i + 1) + 1)$. In addition, the number of *TF*s required by merging two *EF*s is $(\frac{1}{\epsilon}+1)(\log(\epsilon(n_i+n_j))+1)$. As a consequence, the total required memory (the total number of *TF*s) of clustering streaming trajectories using the sliding window model is $O(\frac{k}{\epsilon}\log(\epsilon\lceil\frac{W}{k}\rceil))$.

## 4 Experimental evaluation

In this section, we conduct extensive experiments to evaluate the clustering performance and efficiency of our proposal. First, we utilize TRACLUS [6] as the baseline approach to compare against OCluST. OCluST extracts spatio-temporal characteristics of trajectories at different intervals with very little information loss. TRACLUS clusters over the original trajectory data set, and is regarded as the most effective trajectory clustering algorithm available for a static trajectory data set. Therefore, we choose the clustering result of TRACLUS as a precision evaluation standard. To better perform an accuracy comparison against TRACLUS, we fit a hurricane data set and deer movement data set in a single window. Second, in order to verify the effectiveness and efficiency of OCluST on streaming trajectories, we compare OCluST against two algorithms (TCMM [12] and TSCluWin [21]) on a trajectory data set for taxis. TCMM is a representative incremental trajectory clustering approach that employs a micro- and macro-clustering framework to cluster trajectory data. Unlike OCluST, TCMM does not take the temporal aspects of the trajectories into account, and cannot eliminate obsolete trajectories. Finally, we evaluate traffic conditions on-the-fly by executing the OCluST algorithm on a trajectory data set for taxis.

All code, written in Java, is run on a PC with an Intel Core CPU at 3.1 GHz and 8.00 GB of RAM. The operating system is Windows 8.1. Unless mentioned, the parameters are set below: $\epsilon = 0.5, \gamma = 0.75, \rho = 0.5$.

### 4.1 Data sets

We use four real-world trajectory data sets, including a hurricane track data set, a deer movement data set, and two taxi trajectory data sets. The former two are derived in free space, and the latter two are obtained on a restricted road network.

Just like [6], we choose the Atlantic hurricane track data within the period 1950 to 2004, and extract the latitude and longitude information of hurricanes for the experiment. The deer movement data set contains the radio-telemetry locations of deer in 1995. We extract the $x$ and $y$ coordinates from *Deer*1995 for the experiment.

The taxis' trajectory data sets contain the GPS logs of taxis over a period of three months from October to December 2013, covering the main road networks of Shanghai and Beijing. Each GPS log is represented by a sequence of time-stamped points (latitude and longitude positions).

### 4.2 Effectiveness evaluation

• **Results for hurricane track data** To verify the accuracy of our proposal, we first implement TRACLUS and OCluST on hurricane track data. Both algorithms use the DBSCAN algorithm in different phases. $\min Lns$ and $\epsilon$ are important parameters for DBSCAN, and $\epsilon$ is a neighbor threshold. To differentiate it from the error threshold in our proposal, we use $d$ to denote the neighbor threshold for DBSCAN. Figure 6 shows the clustering result of TRACLUS and the macro-clustering result of OCluST using the optimal parameter values (TRACLUS: $\min Lns = 9$, $d = 130,000$, OCluST: $k = 1000$, $\min Lns = 30$, $d = 320,000$). Thin green lines depict raw trajectories, and thick red lines with arrows represent the extracted clusters (directions marked using arrows), specifically, macro-clusters for OCluST. The clustering results of both algorithms are similar except for a few minor differences. They all capture the moving trends of hurricanes accurately, that is, moving from east to west first and then moving from west to east, occasionally mixed with south-to-north movement. As we can see, four clusters (Fig. 6(a)) are identified by TRACLUS, and three macro-clusters (Fig. 6(b)) are identified by OCluST. In addition, the lengths and locations of representative line segments in both algorithms exist with very few deviations. This is mainly because OCluST executes macro-clustering using the DBSCAN algorithm on a set of representative trajectory segments of micro-clusters, which summarizes the trajectories with very little information loss. TRACLUS executes clustering on trajectory partitions of original trajectories using the DBSCAN algorithm and derives the representative trajectories. It is noted that the clusters obtained by the DBSCAN algorithm are easily affected by the direction and length of core line segments and border line segments. Since the direction and length of the representative line segments in OCluST are different from those of trajectory partitions in TRACLUS, the clustering results in Fig. 6 are quite reasonable.

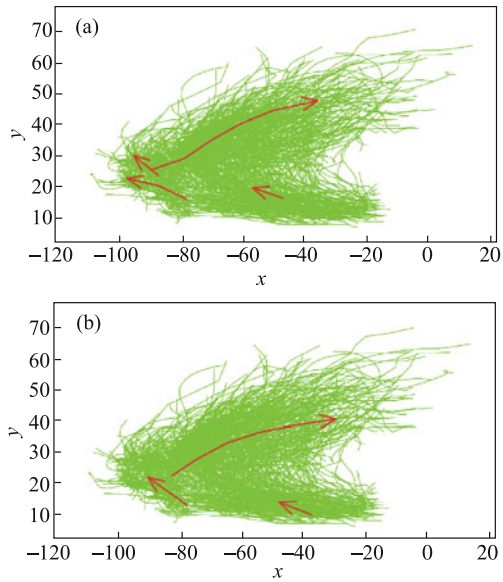


**Fig. 6** Clustering results for hurricane track data. (a) Hurricane-TRACLUS; (b) Hurricane-OCluST

• **Results for deer movements in 1995** In the subsequent effectiveness experiment, we report on the clustering results of TRACLUS and OCluST on deer movements data. Figure 7 shows the clustering results for both algorithms by using the appropriate parameter values (TRACLUS: $\min Lns = 8, d = 149$, OCluST: $k = 300, \min Lns = 100, d = 750$). Also, we use thin green lines to depict raw trajectories and thick red lines to represent the extracted clusters (directions marked using arrows). Like the results for the hurricane track data, except very few differences in the lengths and locations of representative line segments, the clustering results of both algorithms are broadly similar. As illustrated in Fig. 7, on the left side of the middle area, we observe that deer essentially move in different directions, and the movement data are not so dense to form a cluster. Hence, we derive two clusters in the two most dense regions, which match the intuitive clusters.

• **Results for Shanghai taxi trajectory data** For ef-

fectiveness validation purpose, in addition to comparing our proposal with TRACLUS in free space, we conduct evaluations by comparing OCluST with two algorithms (TCMM and TSCluWin) on a restricted road network. The three approaches maintain the same number of micro-clusters. As for the important parameter $d_{\max}$ of TCMM, we set the same value as [12], i.e., $d_{\max} = 800$.

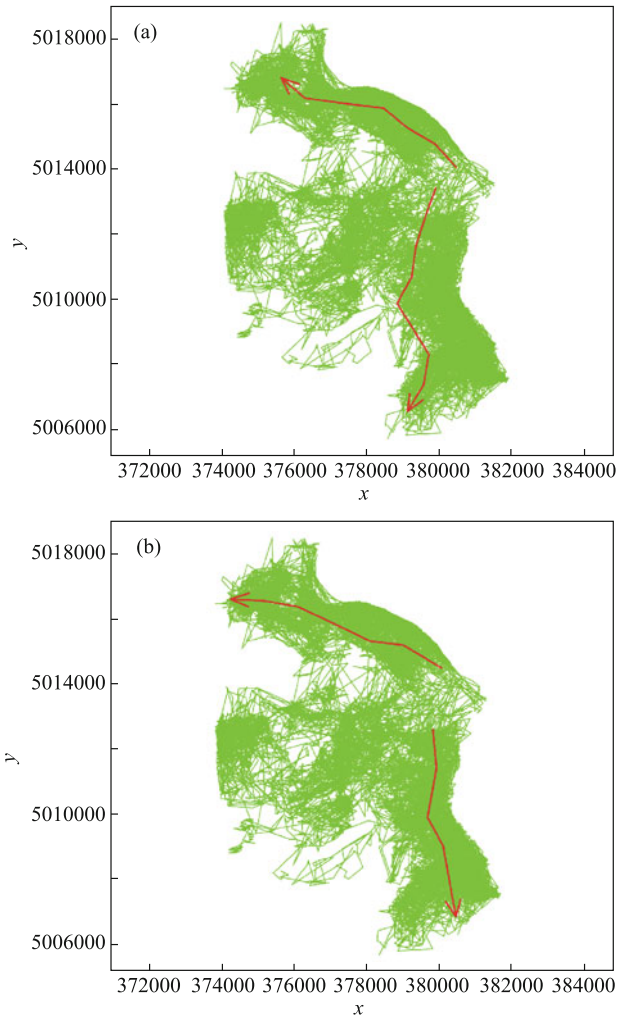


**Fig. 7** Clustering results for Deer1995. (a) Deer1995-TRACLUS; (b) Deer1995-OCluST

Here, we consider the sum of the square distance (SSQ) used in [12,13] as the criterion to evaluate the quality of the clustering results (micro-clusters or macro-clusters). For each cluster $c_i$, we first obtain the SSQ by computing the sum of the square distance between each line segment in $c_i$ (denoted as $L_j^i$, $0 < j \leqslant n_i$) and the representative line segment of $c_i$ (denoted as $c_i.rp$), and then derive the average SSQ by calculating the ratio of SSQ to the number of trajectory line segments (denoted as $n$). Therefore, the average SSQ can be calculated as

$$\text{Average } SSQ = \frac{1}{n}\sum_{i=1}^{k}\sum_{j=1}^{n_i} DL^2(L_j^i, c_i.rp),$$

where $k$ denotes the number of clusters (micro-clusters or macro-clusters). Generally, a smaller average SSQ value means a higher clustering quality.

First, to validate the effectiveness of our proposal, we compare the micro-clustering results obtained by three algorithms (OCluST, TSCluWin, and TCMM) under different window sizes. In terms of the reported average SSQ value, we conclude that OCluST and TSCluWin cluster the streaming trajectories more effectively than TCMM, and OCluST obtains the best clustering results.

Figure 8 shows the average SSQ obtained by three approaches (OCluST, TSCluWin, TCMM) as time progresses when the window size is set to 160,000 and 330,000, respectively. We observe that in all cases, OCluST and TSCluWin behave better (with a smaller average SSQ value) than TCMM because obsolete records are promptly eliminated, and the clustering changes in the most recent records in the current window can be precisely captured by OCluST and TSCluWin. Thus, 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 increase on the boundary rather than be split into multiple small micro-clusters. Additionally, OCluST invariably obtains better outcomes than TSCluWin, as illustrated in Fig. 8. With the dramatic increase of positional data, TSCluWin easily obtains larger micro-clusters than OCluST because longer representative line segments are easily derived by TSCluWin approach when the positional data has great distances between corresponding starting positions or ending positions. As a result, uneven clustering results drastically degrade the overall performance of micro-clustering, as illustrated in Fig. 8(a). Furthermore, as data continue to flow in, OCluST approach shows better clustering efficacy with the use of more effective methods for extracting representative line segments. At the same time, the clustering efficacy is less influenced by larger window sizes, as illustrated in Fig. 8(b).

### 4.3 Efficiency evaluation

• **Execution time evaluation** We assess the efficiency of our proposal by comparing OCluST and TSCluWin with TCMM when dealing with streaming trajectories. Figure 9 shows a per-record processing time comparison (expressed in seconds) among three methods. The OCluST curves nearly

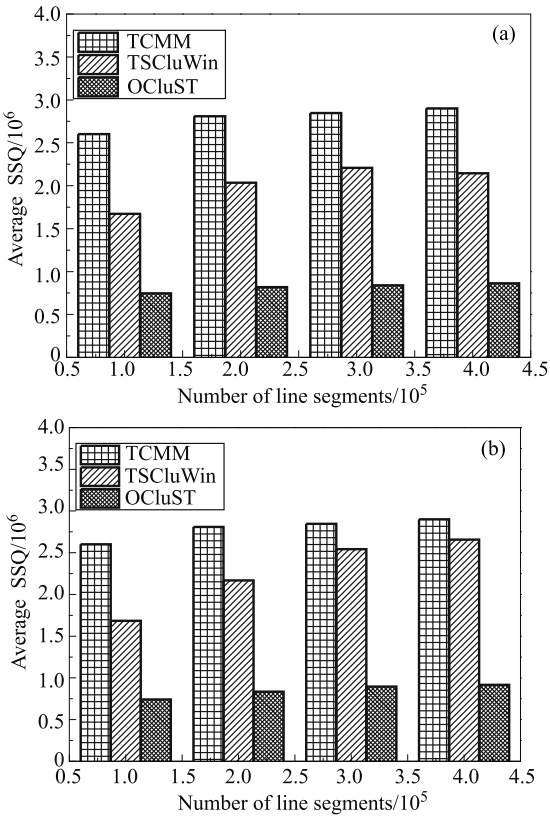


**Fig. 8** Quality comparison. (a) $W = 160,000$; (b) $W = 330,000$

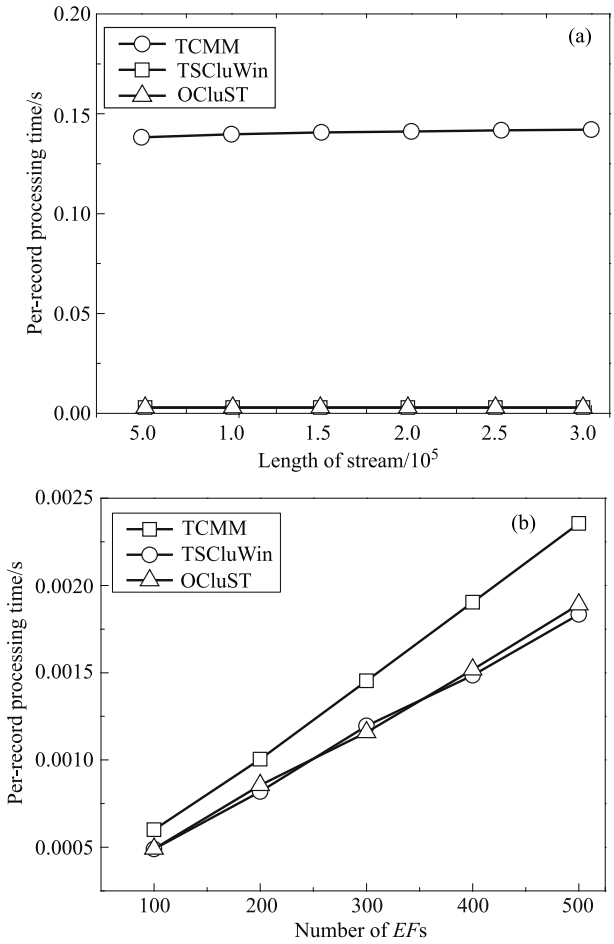


**Fig. 9** Execution time comparison. (a) Execution time vs. length of stream; (b) Execution time vs. number of *EF*s

overlap the TSCluWin curves. As shown in Fig. 9(b), the per-record processing time of OCluST and TSCluWin fluctuates smoothly and keeps superior to that processed by TCMM with the progression of a trajectory stream. TCMM takes 0.13s to obtain a cluster, while OCluST and TSCluWin only take around 0.008s. The faster processing rate of our proposal is because micro-clustering is executed on the original trajectory data (without disregarding any trajectory points). By contrast, TCMM needs to partition trajectories using MDL method before the micro-clustering phase, which consumes additional waiting time for partitioning accumulated trajectory data into sub-trajectories.

Additionally, to test the algorithm's robustness in the presence of uncertainty, we study the sensitivity of our proposal to parameter $k$. Figure 9(b) shows the per-record processing time when the number of micro-clusters is varied. Since the distance computation cost of finding the most similar micro-cluster for an incoming line segment keeps growing as the number of clusters increases, all the approaches scale linearly with the number of micro-clusters. Nevertheless, with an increase in the class number, the execution overheads of OCluST and TSCluWin grow more slowly than that of TCMM, and are less affected by the parameter $k$.

• **Memory usage evaluation** One important efficiency characteristic for the streaming algorithms is the memory usage. As previously mentioned, *TF* and *EF* synopsis structures extract and maintain the spatio-temporal characteristics of micro-clusters at different intervals, which effectively compresses the size of the data needed to be stored. Therefore, we evaluate the memory usage of two algorithms mainly by using the memory space usage of the synopsis structures (*TF*) without taking into account the other issues. Since this type of entity (*TF* synopsis structure) defined in the OCluST and TSCluWin methods needs almost the same memory space, we utilize the number of entities (*TF*) to estimate the memory usage of both algorithms. Figure 10(a) shows the memory footprint (in the number of entities) of OCluST and TSCluWin when the window size $W$ is set to 160,000 and 330,000, respectively. For $W = 160,000$, as the number of trajectories increases from 100,000 to 400,000, the memory usage of both methods fluctuates with the progression of the trajectory stream, and OCluST requires less memory than TSCluWin. This is based on the fact that a more bal-

anced clustering result is easily obtained by OCluST than TSCluWin. By contrast, for $W = 330,000$, the memory usage of TSCluWin outperforms OCluST at first, and then equals OCluST when the number of incoming records is beyond 240,000. This is because the maintenance of the most similar *EF* for $Z_{list}$ in OCluST is more complicated than that in TSCluWin. A larger window size leads to more influenced *EF*s need to update their most similar *EF*s. However, when the number of trajectories is larger than 240,000, the number of *TF*s drops with the elimination of expired records (as illustrated in Section 3). Both methods require almost the same memory usage.

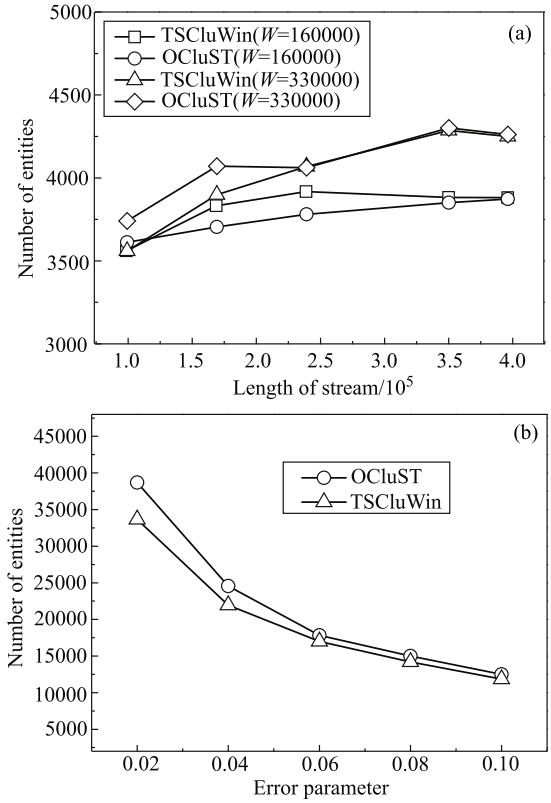


**Fig. 10** Memory usage comparison. (a) Memory usage vs. length of stream; (b) Memory usage vs. threshold $\epsilon$

Moreover, we study the effect of input parameter $\epsilon$ on memory usage. Figure 10(b) shows the memory usage of OCluST and TSCluWin when the window size is set to 250,000 by tuning the value of parameter $\epsilon$. OCluST maintains a slightly larger memory footprint than TSCluWin. In addition, when the value of $\epsilon$ increases from 0.02 to 0.1, the memory usage of both approaches decreases significantly. This is because $\epsilon$ determines the number of expired records within the time window. Specifically, in the current window, with the increment of $\epsilon$, more obsolete records are eliminated, and fewer TFs are stored in memory.

### 4.4 Application to real-world problem: real-time traffic information evaluation

In order to further assess the performance of OCluST on real-world problems, we apply OCluST to a taxi trajectory data set to derive real-time traffic information for an urban road network. Although there are specific pervasive techniques and on-road (fixed) sensors for estimating traffic situations, e.g., magnetometers, visual cameras, and inductive loops built into the road surface, they are prone to error and are limited to critical portions of the arterial network owing to their high costs of implementation and maintenance. This requires some new ways to collect traffic data for evaluating road situations. Since taxis with in-vehicle GPS devices travel along the entire road network, their trajectories typically cover much wider areas, and hence they can be treated as mobile probes to measure traffic states. As a complementary solution to fixed sensors, this opens the door to use our proposal for deeply understanding road network traffic, especially in areas where no sufficient infrastructure to estimate traffic has or can be deployed. Among the existing traffic estimation methods, the aggregate-based method [22,23] is widely used to estimate the road conditions of each link (i.e., a unidirectional road segment) owing to its simplicity. However, it aims at generating the aggregate values (e.g., average speed and maximum speed) of each road link, and cannot reveal similar moving behavior of objects that drive on each road segment or on wider road regions. Thus, to indicate the roads where vehicular traffic is most concentrated, and to estimate the mean speed of each road segment in real time, we use OCluST instead of the aggregate-based method to cluster and discover the common behavior of each cluster.

• **Indications of high-vehicular-density route distribution in Shanghai** We first implement OCluST approach on Shanghai's taxi trajectory data set to indicate the roads where vehicular traffic is most concentrated during a specified time period. The clustering result of trajectories on a randomly selected region is visualized in Fig. 11. Figure 11(a) shows the movement distribution of taxi traces during the period (from 10:00 to 10:30 a.m.) on October 7, which involves 477,810 trajectory line segments (in green). Let *MinLns* be set to 2000 and $d$ to 730. We implement OCluST algorithm on trajectories within the intervals [10:00, 10:10] and [10:10, 10:20] separately. These are portions of data shown in Fig. 11(a). Then, we derive 300 micro-clusters (in blue) on trajectory segments within each interval, as shown in Figs. 11(b) and 11(c). It is observed that the micro-clustering results within the two intervals can capture most traffic flows in Fig.

11(a), but with a few differences, which does accord with the actual dynamic traffic status. Additionally, since micro-clustering is maintained on raw trajectory line segments, the obtained representative trajectory of each cluster behaves in a short route form, and hence the micro-clusters are discrete on the map. The micro-clustering result indicates the road segments where vehicular traffic is the most concentrated. Given the time period (from 10:00 to 10:30 a.m.) on October 7, we execute macro-clustering using DBSCAN algorithm. Then, small micro-clusters are derived within three intervals ([10:00, 10:10], [10:10, 10:20], and [10:20, 10:30]), which are merged into seven larger macro-clusters. As illustrated in Fig. 11(d), these macro-clusters describe seven high-vehicular-density routes, which also reflect the traffic continuity in the road network. Essentially, the seven routes on the map indicate seven hot regions, including Shanghai Railway Station, Shanghai Museum, West Nanjing Road, and Caoyang Road. Moreover, it can be observed that some traces originally exist in Fig. 11(a) but disappear in Fig. 11(d) because there are insufficient participating trajectory line segments to be filtered out. The real-time traffic information for high-vehicular-density route distribution can be delivered to road users for traffic management.

• **Estimate the mean speed of each road segment of Beijing in real time** We also attempt to use OCluST approach to estimate the traffic situation in Beijing on October 9. As mentioned earlier, the clustering of streaming trajectories is more flexible in evaluating real-time traffic conditions than the aggregate-based method. For instance, we can estimate a varied range of road region traffic situations by using different


**Fig. 11** Indications of high-vehicular-density route distribution in Shanghai. (a) Movement distribution within (10:00–10:30); (b) Micro-clustering within (10:00–10:10); (c) Micro-clustering within (10:10–10:20); (d) Macro-clustering within (10:00–10:30)

distance thresholds in clustering. In our experiment, the average speed is employed to estimate the traffic situation because other information can be derived by speed (e.g., estimated travel time). We treat the average speed of each macro-cluster as the mean speed of a road segment where a macro-cluster is located. By setting speed thresholds, we empirically divide the traffic conditions into three classes: traffic jams with speeds lower than 28.8km/h (in red), less congested traffic with speeds within a range [28.8km/h,54km/h] (in yellow), and smooth traffic with speeds beyond 54km/h (in green). Figure 12 shows the macro-clustering results for four intervals (8:00–8:30, 10:00–10:30, 14:00–14:30, and 17:00–17:30) on October 9, 2013. As shown in Fig. 12(a), for [8:00–8:30], the distribution of congestion is from the outside to the inside of the city. Congested roads are mainly concentrated in the south and north directions of East 2nd Ring Road, West 2nd Ring Road, East 3rd Ring Road, and most highways entering Beijing (in red on the map). Traffic moves slowly in the Zhongguancun Bridge area, Hangtianqiao area, Yongding Road, Wanquanhe Road, and Yuanmingyuan West Road (in yellow on the map). During the interval [17:00–17:30], as illustrated in Fig. 12(d), there is serious congestion in the north-to-south direction of East and West Second Ring Roads, as well as the north-to-south directions of East and West Third Ring Roads. Vehicle travel is more concentrated in the famous business circles and surrounding Catering areas, e.g., the Dongzhimen, Xizhimen, CBD, Financial Street, and Zhongguancun regions (in red on the map). Moreover, the traffic crawls in the regions near Beijing West Railway Station and Beijing South Railway Station (in yellow on the map). During the other two intervals (10:00–10:30 and 14:00–14:30), congestion is not serious and traffic is rela-

**Fig. 12** Estimating mean speed of road segments in Beijing. (a) 8:00–8:30; (b) 10:00–10:30; (c) 14:00–14:30; (d) 17:00–17:30

tively smooth. The vehicle velocity changes that were derived from taxi trajectories can exhibit peaks in the morning and afternoon rush hours, and basic impartial traffic pressure during off-peak hours. As a result, the clustering results of OCluST algorithm can effectively evaluate traffic conditions in real-time.

## 5 Related work

In this section, we review existing work on data stream clustering and trajectory clustering, and then cover a continuous query over a trajectory stream. We end by introducing a few research studies devoted to the clustering of streaming trajectories.

• **Clustering of data stream** Traditional clustering approaches include partition-based methods (e.g., K-means, [24]), hierarchical-based methods (e.g., BIRCH, [25]), density-based methods (e.g., DBSCAN, [7]) and grid-based methods (e.g., STING, [9]). All of the above methods need to visit the data set more than once. Thus, they are unsuitable for streaming scenarios with real-time response requirements. Babcock et al. studied the clustering issue using the sliding-window model with a focus on theoretical bound analysis [26]. Aggarwal et al. developed CluStream algorithm to cluster large evolving data streams based on a pyramid model [13]. Aggarwal et al. further proposed UMicro algorithm to deal with uncertain data streams [27]. Zhou et al. presented SWClustering algorithm to track the evolution of clusters using the sliding-window model by using a novel synopsis data structure called EHCF [28]. Jin et al. proposed the cluUS algorithm to cluster uncertain data streams using the sliding-window model [29]. However, none of the above methods can deal with trajectory data directly owing to their different scenarios. In such scenarios, each tuple in the data stream is an entry, while each tuple in streaming trajectories is only part of an entry.

• **Clustering on static trajectory data** There exist salient accomplishments on clustering static trajectory data sets, including road-network unaware sets [6,8] and road-network aware sets [20,30,31]. Gaffney et al. treated the entire trajectory as a basic unit and introduced fundamental principles of clustering trajectory based on the probabilistic mixture regression model [8]. Lee et al. presented a partition-and-group framework (TRACLUS) to derive common sub-trajectories [6]. Since the MDL method used in partitioning the trajectory suffers from high computation overhead, it is not suited for online clustering trajectories with limited memory resources. Many research studies address road-network scenarios [20,30,31]. Won et al. presented a similarity measurement by considering the total length of matched road segments, and proposed a clustering algorithm by adjusting the FastMap and hierarchical clustering schemes [30]. Roh et al. proposed a distance measure that considers the spatial proximity of vehicle trajectories on a road network, and presented a neighbor-based clustering approach called NNCluster [20]. Han et al. proposed a road-network aware clustering method called NEAT, which considers traffic locality characterized by the spatial constraints of a road network, the traffic flow among consecutive road segments, and the flow-based density [31]. Nevertheless, the abovementioned works require clustering on stored trajectory data; thus, they cannot be applied to streaming trajectories directly.

• **Continuous query over trajectory stream** There also exist some approaches that are to some degree orthogonal to our work but deserve to be mentioned. Various techniques for trajectory simplification [6] or trajectory compression [14] have been studied in real-time trajectory tracking [32], but they refer to the problem of minimizing the amount of position data that are communicated and stored. Owing to high computational overhead to attain the optimal results, these approaches are unsuitable for online clustering rapidly changing stream data in limited memory. In addition, more recent achievements have been reported for continuous query processing over trajectory streams [33–37]. Nehme et al. proposed SCUBA to optimize the execution of continuous queries on spatio-temporal data streams by utilizing motion clustering [33]. Sacharidis et al. proposed a framework for the online maintenance of hot motion paths in order to detect frequently traveled trails of numerous moving objects [34]. Zheng et al. presented a method to discover closed gathering patterns from a large trajectory data set [35]. Tang et al. studied the problem of incrementally discovering traveling companions from streaming trajectories by a data structure termed the traveling buddy [36]. Li et al. proposed a group discovery framework that satisfies requirements including sampling independence, density connectedness, trajectory approximation, and online processing [37].

• **Clustering of streaming trajectories** Trajectory clustering for static data sets [6,8] scarcely considers maintaining clusters incrementally. Jensen et al. exploited an incrementally maintained clustering feature CF, and proposed a scheme for the continuous clustering of moving objects [10]. Li et al. presented the concept of a Moving Micro-Cluster to catch the regularities of moving objects [11]. Nevertheless, they highlighted incrementally clustering moving objects rather than trajectories. Li et al. proposed an incremen-

tal trajectory clustering framework called TCMM, which includes micro-clusters maintenance by simplifying new trajectories into directed line segments, and macro-clusters generated by clustering the micro-clusters [12]. While in the micro-clustering phase, TCMM has to accumulate sufficient incoming positional data to obtain the simplified sub-trajectories by using MDL method. In addition, owing to the effect of obsolete data, along with the process of continuously absorbing more records and merging the most similar pairs of micro-clusters, the centers of micro-clusters will shift gradually, which leads to concept drift and thus degrades the effectiveness of the resulting clusters. In general, incremental clustering approaches barely consider the temporal aspects of the trajectories and cannot scale up to mine massive trajectory streams.

Aiming at the requirement of clustering trajectory big data, Deng et al. presented a scalable density-based clustering algorithm called Tra-POPTICS, and parallelized it with the Hyper-Q feature of Kelper GPU and massive GPU threads [38]. Costa et al. proposed a framework (Lifting and Fourier Transforms) that pre-elaborates original trajectories by using non-separable Fourier transforms [39]. Yu et al. proposed CTraStream for clustering trajectory data streams, including online line-segment stream clustering and an update process for closed trajectory clusters based on a TC-Tree index [40]. This attempts to extract the patterns online, similar to a convoy pattern [41]. Our previous work [21] proposed a two-phase framework to cluster a trajectory stream using the sliding-window model, called TSCluWin. It is capable of capturing the clustering changes in a certain temporal window while eliminating the influence of expired data. This paper builds on this concept but differs in the following respects. First, we exploit a new type of synopsis data structure to summarize the spatio-temporal clustering features of trajectories at different instants. Second, we employ an online approach to cluster streaming trajectories (OCluST) on the basis of such synopses. Third, we conduct performance studies on real data sets by comparing OCluST with TRACLUS, TCMM, and TSCluWin algorithms. Finally, we use OCluST algorithm to estimate real-time traffic conditions in urban road networks, specifically, by extracting high-vehicular-density routes and assessing the traffic pressure according to the velocity changes in the clustering results.

## 6 Conclusion and future work

In this paper, we proposed an online algorithm called OCluST to cluster evolving streaming trajectories using the sliding-window model. It consists of two components: a micro-clustering component that summarizes trajectory line segments in the current window, and a macro-clustering component that reclusters the previously extracted summaries according to the user's request. Specifically, we define two novel synopsis data structures (TF and EF) to represent the spatio-temporal clustering characteristics of the stream data in memory, and track the latest cluster changes of the trajectory stream in real time. By conducting extensive experiments on real-world data sets, we compare OCluST to three other algorithms (TRACLUS, TCMM, and TSCluWin) in terms of effectiveness and efficiency. A theoretical analysis and comprehensive experimental results demonstrate that our proposal is of high quality, requires little memory, has a fast processing rate in coping with streaming trajectories, and outperforms the baseline approach.

Trajectory data in real applications is generally collected in a distributed fashion. With a substantial increment in the trajectory stream data, a distributed clustering approach is required to meet the rising requirements of analyzing huge amounts of data. Therefore, in our future work, we would like to extend OCluST to a distributed solution based on a distributed computing platform in order to improve the efficiency of processing massive volumes of streaming trajectories and to provide real-time clustering results.

**Acknowledgements** Our research was supported by the National Key Research and Development Program of China (2016YFB1000905), the National Natural Science Foundation of China (NSFC) (Grant Nos. 61702423, 61370101, 61532021, U1501252, U1401256 and 61402180), Natural Science Foundation of the Education Department of Sichuan Province (17ZA0381 and 13ZA0015), China West Normal University Special Foundation of National Programme Cultivation (16C005), and Meritocracy Research Funds of China West Normal University (17YC158).

## References

1. Pan B, Zheng Y, Wilkie D, Shahabi C. Crowd sensing of traffic anomalies based on human mobility and social media. In: Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems. 2013, 334–343
2. Liu H P, Jin C Q, Zhou A Y. Popular route planning with travel cost estimation. In: Proceedings of International Conference on Database Systems for Advanced Applications. 2016, 403–418
3. Chen C, Chen X, Wang Z, Wang Y S, Zhang D Q. ScenicPlanner: planning scenic travel routes leveraging heterogeneous user-generated digital footprints. Frontiers of Computer Science, 2017, 11(1): 61–74
4. Duan X Y, Jin C Q, Wang X L, Zhou A Y, Yue K. Real-time personalized taxi-sharing. In: Proceedings of International Conference on Database Systems for Advanced Applications. 2016, 451–465
5. Wu H, Tu C C, Sun W W, Zheng B H, Su H, Wang W. GLUE: a

parameter-tuning- free map updating system. In: Proceedings of the 24th ACM International Conference on Information and Knowledge Management. 2015, 683–692

6. Lee J G, Han J W, Whang K Y. Trajectory clustering: a partition-and-group framework. In: Proceedings of ACM SIGMOD International Conference on Management of Data. 2007, 593–604
7. Ester M, Kriegel H P, Sander J, Xu X W. A density-based algorithm for discovering clusters in large spatial databases with noise. In: Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining. 1996, 226–231
8. Gaffney S, Smyth P. Trajectory clustering with mixtures of regression models. In: Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 1999, 63–72
9. Wang W, Yang J, Muntz R R. STING: a statistical information grid approach to spatial data mining. In: Proceedings of the 23rd International Conference on Very Large Data Bases. 1997, 186–195
10. Jensen C S, Lin D, Ooi B C. Continuous clustering of moving objects. IEEE Transactions on Knowledge & Data Engineering, 2007, 19(9): 1161–1174
11. Li Y F, Han J W, Yang J. Clustering moving objects. In: Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. 2004, 617–622
12. Li Z H, Lee J G, Li X L, Han J W. Incremental clustering for trajectories. In: Proceedings of the 15th International Conference on Database Systems for Advanced Applications. 2010, 32–46
13. Aggarwal C C, Han J W, Wang J Y, Yu P S. A framework for clustering evolving data streams. In: Proceedings of the 29th International Conference on Very Large Data Bases. 2003, 81–92
14. Hönle N, Großmann M, Reimann S, Mitschang B. Usability analysis of compression algorithms for position data streams. In: Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems. 2010, 240–249
15. Datar M, Gionis A, Indyk P, Motwani R. Maintaining stream statistics over sliding windows. SIAM Journal on Computing. 2002, 31(6): 635–644
16. Chu S, Keogh E J, Hart D M, Pazzani M J. Iterative deepening dynamic time warping for time series. In: Proceedings of the 2nd SIAM International Conference on Data Mining. 2002, 195–212
17. Vlachos M, Gunopulos D, Kollios G. Discovering similar multidimensional trajectories. In: Proceedings of the 18th International Conference on Data Engineering. 2002, 673–684
18. Chen L, Ng R T. On the marriage of Lp-norms and edit distance. In: Proceedings of the 30th International Conference on Very Large Data Bases. 2004, 792–803
19. Chen L, Özsu M T, Oria V. Robust and fast similarity search for moving object trajectories. In: Proceedings of ACM SIGMOD International Conference on Management of Data. 2005, 491–502
20. Roh G, Hwang S. NNCluster: an efficient clustering algorithm for road network trajectories. In: Proceedings of International Conference on Database Systems for Advanced Applications. 2010, 47–61
21. Mao J L, Song Q G, Jin C Q, Zhang Z G, Zhou A Y. TSCluWin: trajectory stream clustering over sliding window. In: Proceedings of International Conference on Database Systems for Advanced Applications. 2016, 133–148
22. Zhang J, Xu J, Liao S S. Aggregating and sampling methods for processing GPS data streams for traffic state estimation. IEEE Transactions on Intelligent Transportation Systems, 2013, 14(4): 1629–1641
23. Castro P S, Zhang D Q, Li S J. Urban traffic modelling and prediction using large scale taxi GPS traces. In: Proceedings of International Conference on Pervasive Computing. 2012, 57–72
24. Lloyd S P. Least squares quantization in PCM. IEEE Transactions on Information Theory, 1982, 28(2): 129–136
25. Zhang T, Ramakrishnan R, Livny M. BIRCH: an efficient data clustering method for very large databases. In: Proceedings of ACM SIGMOD International Conference on Management of Data. 1996, 103–114
26. Babcock B, Datar M, Motwani R, O'Callaghan L. Maintaining variance and k-medians over data stream windows. In: Proceedings of the 22nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems. 2003, 234–243
27. Aggarwal C C, Yu P S. A framework for clustering uncertain data streams. In: Proceedings of IEEE International Conference on Data Engineering. 2008, 150–159
28. Zhou A Y, Cao F, Qian W N, Jin C Q. Tracking clusters in evolving data streams over sliding windows. Knowledge and Information Systems, 2008, 15(2): 181–214
29. Jin C Q, Yu J X, Zhou A Y, Cao F. Efficient clustering of uncertain data streams. Knowledge and Information Systems, 2014, 40(3): 509–539
30. Won J I, Kim S W, Baek J H, Lee J Trajectory clustering in road network environment. In: Proceedings of IEEE Symposium on Computational Intelligence and Data Mining. 2009, 299–305
31. Han B, Liu L, Omiecinski E. Road-network aware trajectory clustering: integrating locality, flow, and density. IEEE Transactions on Mobile Computing, 2015, 14(2): 416–429
32. Lange R, Dürr F, Rothermel K. Efficient real-time trajectory tracking. The VLDB Journal, 2011, 20(5): 671–694
33. Nehme R V, Rundensteiner E A. SCUBA: scalable cluster-based algorithm for evaluating continuous spatio-temporal queries on moving objects. In: Proceedings of the 10th International Conference on Advances in Database Technology. 2006, 1001–1019
34. Sacharidis D, Patroumpas K, Terrovitis M, Kantere V, Potamias M, Mouratidis K, Sellis T. On-line discovery of hot motion paths. In: Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology. 2008, 392–403
35. Zheng Y, Yuan N J, Zheng K, Shang S. On discovery of gathering patterns from trajectories. In: Proceedings of IEEE International Conference on Data Engineering. 2013, 242–253
36. Tang L A, Zheng Y, Yuan J, Han J W, Leung A, Hung C C, Peng W C. On discovery of traveling companions from streaming trajectories. In: Proceedings of the 28th IEEE International Conference on Data Engineering. 2012, 186–197
37. Li X H, Ceikute V, Jensen C S, Tan K L. Effective online group discovery in trajectory databases. IEEE Transactions on Knowledge & Data Engineering, 2013, 25(12): 2752–2766
38. Deng Z, Hu Y Y, Zhu M, Huang X H, Du B. A scalable and fast OPTICS for clustering trajectory big data. Cluster Computing, 2015, 18(2): 549–562
39. Costa G, Manco G, Masciari E. Dealing with trajectory streams by clustering and mathematical transforms. Journal of Intelligent Information Systems, 2014, 42(1): 155–177
40. Yu Y W, Wang Q, Wang X D, Wang H, He J. Online clustering for trajectory data stream of moving objects. Computer Science & Infor-

mation Systems, 2013, 10(3): 1293–1317

41. Jeung H, Yiu M L, Zhou X F, Jensen C S, Shen H T. Discovery of convoys in trajectory databases. Proceedings of the VLDB Endowment, 2008, 1(1): 1068–1080

Jiali Mao is an associate professor at China West Normal University, China. She is currently working toward the PhD degree in the School of Data Science and Engineering, East China Normal University, China. Her current research interests include big data analysis and location-based services.

Qiuge Song received her bachelor's degree in computer science and technology from Nankai University, China in 2014. She is a graduate student in the School of Software Engineering, East China Normal University, China. Her current research interests include data mining and location-based services.

Cheqing Jin is a professor of computer science at East China Normal University, China. He received the Excellent Young Teacher Award from Fok Ying Tung Education Foundation. His main research interests include streaming data management, location-based services, uncertain data management, data quality, and database benchmarking.

Zhigang Zhang is currently working toward the PhD degree at the School of Data Science and Engineering, East China Normal University, China. His research interests include location-based services, spatio-temporal data management, and distributed computing.

Aoying Zhou is a professor of computer science at East China Normal University (ECNU), China, as well as the dean of the School of Data Science and Engineering (DaSE), ECNU. His research interests include web data management, data management for data-intensive computing, in-memory cluster computing, benchmarking for big data, and performance.