Appl Intell (2018) 48:2661–2680
https://doi.org/10.1007/s10489-017-1104-z

Trajectory outlier detection approach based on common slices sub-sequence

Qingying Yu[1,2] · Yonglong Luo[1,2] · Chuanming Chen[1,2] · Xiaohan Wang[1,2]

Published online: 18 December 2017

Abstract Trajectory outlier detection is one of the most popular trajectory data mining topics. It helps researchers obtain a lot of valuable information that can be used as important guidance in monitoring and forecasting. Existing methods have difficulty in detecting the outlying trajectories with continuous multi-segment exception. To address the problem, in this paper, we propose a novel trajectory outlier detection algorithm based on common slices sub-sequence (TODCSS). For each trajectory, the direction-code sequence is firstly calculated based on the direction of each trajectory segment. Secondly, the corresponding sequence consisting of trajectory slices is obtained by inflection point segmentation. And then, the common slices sub-sequences between two trajectories are found to measure their distance. Finally, the slice outliers and trajectory outliers are detected based on the new CSS distance calculation. Both the intuitive visualization presentation and the experimental results on real Atlantic hurricane dataset, real-life mobility trajectory dataset of taxis in San Francisco and synthetic labeled dataset show that the proposed TODCSS algorithm effectively detects slice and trajectory outliers, and improves accuracy and stability in trajectory outlier detection.

Keywords Trajectory outlier detection · Trajectory segment · Trajectory slice · Common slices sub-sequence (CSS)

1 Introduction

With the rapid development of various GPS-enabled devices and wireless communication technologies, large amounts of trajectory data of moving objects have been generated and collected. Examples include moving data of mobile users, vehicles, animals and hurricanes [1, 2]. Analysis on such data can help researchers obtain a lot of valuable information, such as hot spots, interest patterns, location predication and other related implied facts [3–5]. Therefore, there is an increasing interest to perform trajectory pattern mining, in which trajectory outlier detection (TOD) is one of the most popular research topics [6–9].

It is well known that an outlier means a data object that is grossly different from or inconsistent with the remaining set of data [10]. It is a very rare pattern that may indicate an abnormal event. Therefore, outlier detection is an important data analysis task [11]. A trajectory outlier (also known as an outlying trajectory) is a trajectory that has a great difference with most other trajectories in a certain time interval based on some similarity evaluation mechanism. TOD belongs to the category of spatio-temporal outlier detection because trajectory has spatial and temporal feautures [12, 13]. The detection results can help identify suspicious activities of moving objects and therefore be used in many applications, such as severe weather prediction, security surveillance, and

✉ Yonglong Luo
ylluo@ustc.edu.cn

Qingying Yu
ahnuyuq@mail.ahnu.edu.cn

Chuanming Chen
ccm_0@163.com

Xiaohan Wang
hanxiaoahnu@sina.com

[1] School of Mathematics and Computer Science, Anhui Normal University, Wuhu 241003, China

[2] Anhui Provincial Key Laboratory of Network and Information Security, Wuhu 241003, China

intelligent transportation and scientific studies [14–16]. As Chen et al. proposed in Literature [17], a tropical cyclone system or a hurricane can be considered an abnormal activity of the atmosphere system. Moreover, the detection and removal of trajectory outliers are significant for improving the efficiency of trajectory similarity clustering algorithm.

Example 1 Consider the trajectory dataset in Fig. 1a. It is obvious that the thick portion of the trajectory T_6 is quite different from the other trajectories. That is, it is an outlying sub-trajectory in this dataset. However, the methods proposed in [18, 19] treating the whole trajectory as a processing unit cannot detect this unusual behavior since the differences may be homogenized. Thus, some important information is possibly missed. To the best of our knowledge, TRAOD method [1] addresses this problem based on a partition-and-detect framework for the first time. In the next year, Bu et al. [20] also proposed another method that divides a given trajectory into equal sized segments.

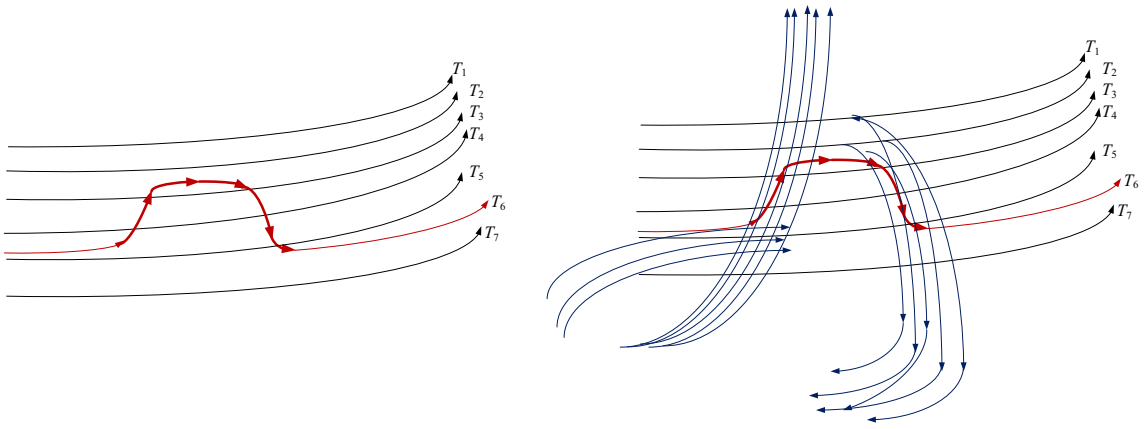
Example 2 Consider the trajectory dataset in Fig. 1b. The outlying sub-trajectory in T_6 cannot be detected by TRAOD because each subpart of the sub-trajectory has enough close neighbors. However, the whole trajectory T_6 is obviously an outlier in such a dataset. The TPRO (time-dependent popular routes based trajectory outlier detection) algorithm [21] takes this problem into account and compares each trajectory with its related conventional routes to find out all the outliers in the historical trajectory dataset. But TPRO algorithm processes each group of trajectories with the same source and destination and detects anomalous trajectories based on the top-k most popular routes.

On the one hand, traditional outlier detection methods dealing with point data objects cannot be directly used in trajectory outlier detection because trajectory data object is significantly different with data point object. On the other hand, existing TOD methods have difficulty in solving the problem pointed out in Fig. 1. Therefore, a powerful TOD algorithm is needed urgently.

In this paper, we propose a new trajectory outlier detection method based on direction sequence of trajectory slices partitioning. For each trajectory, we firstly construct its direction-code sequence based on the direction features of trajectory segments. Secondly, the whole trajectory is partitioned into several trajectory slices. In each slice, the direction-code value is averaged. Thirdly, we detect outlying slices and trajectories based on homogeneous slice distance and common slices sub-sequence.

To sum up, this paper makes the following contributions:

1) *Efficient Trajectory Partition*. A novel and efficient trajectory partition method that holds the potential features is presented, in order to benefit trajectory similarity evaluation and trajectory outlier detection. The method is based on the direction-code sequence and the slice region.
2) *Common slices sub-sequence*. A new conceptual model named as common slices sub-sequence (abbr. CSS) is proposed, which is used to measure the similarity between any two trajectories.
3) *Trajectory outlier detection based on common slices sub-sequence*. A trajectory outlier detection method based on the distance between common slices sub-sequences is proposed. It can address the the problem that an outlying trajectory with continuous multi-segment exception cannot be detected if each segment has enough closed neighbors. Just as is shown in Fig. 1b.
4) *Experimental Study*. The proposed algorithm is tested on real Atlantic hurricane tracking dataset, real-life mobility trajectory dataset and synthetic labeled dataset. The



(a) outlying sub-trajectory can be detected by TRAOD

(b) outlying sub-trajectory cannot be detected by TRAOD

Fig. 1 Two scenarios of outlying sub-trajectory

results show that our algorithm effectively detects the trajectory outliers from a trajectory dataset.

The rest of this paper is organized as follows. In Section 2, we introduce the related work. Section 3 introduces a novel trajectory model. In Section 4, we introduce the generation process of trajectory slices. Section 5 presents a trajectory outlier detection method based on common slices subsequence. Experimental results and analysis are presented in Section 6. Section 7 concludes the paper and provides future research directions.

2 Related work

Recently, much has been achieved in TOD research. In general, the existing trajectory outlier detection algorithms can be classified into two categories based on research content.

The first class focuses on the spatial positions of trajectories. For example, Knorr et al. [18] proposed a method that firstly extracts the location feature vector of trajectory and then uses conventional outlier detection methods to detect outlying trajectories. However, the anomalies of some sub-trajectories may be homogenized and cannot be detected. As shown in Fig. 1a, there is an outlying sub-trajectory in T_6, but the local abnormal characteristics in the overall feature vector cannot be reflected. Li et al. [19] proposed a framework named ROAM (Rule- and Motif-based Anomaly Detection in Moving Objects). In ROAM, discrete pattern fragments (called motifs) are used to represent a trajectory. And the relevant features are extracted to form a hierarchical feature space. They also developed a rule-based classifier to detect abnormal traces. The classification method depends on the labeled training dataset, which cannot be easily obtained. Guan et al. [22] proposed a trajectory outlier detection algorithm relaxing the temporal requirements. That is, the trajectory is outlier only when it do not follow the path that most trajectories followed. Masciari [23] defined a trajectory outlier detection method based on multidimensional Fourier Analysis in order to catch "structural" difference between trajectories. In this method, the speed and time information are disregarded because its aim is to discover unusual structures in trajectory dataset. Anagnostopoulos et al. [4] proposed a spatial variance reduction system to address the problem of noisy motion patterns in trajectory classification. In this system, a distance metric is proposed to achieve fast similarity assessment between two trajectories. It deals with trajectories over $\mathbb{R}^2$.

To detect outlying sub-trajectories, Lee et al. [1] proposed a trajectory outlier detection algorithm TRAOD based on a partition-and-detection framework. First, trajectories are partitioned based on coarse and fine granularity. Then, in order to detect trajectory outliers, each part is compared with its neighbors using a hybrid of the distance-based and density-based approaches. But this method only takes into account the spatial location information, without considering the time information. In addition, the partition is constrained by the base unit. Zhang et al. [24] grouped the trajectories according to the same "start point - end point", and proposed iBAT (Isolation-Based Anomalous Trajectory) algorithm to identify the "less and different" trajectory outliers based on the property that an outlier is susceptible to be isolated. However, it is assumed that trajectories are in the same time period, without analysis in depth. Mohamad et al. [25] proposed that a trajectory is abnormal if it has a sudden speed or direction change. Being similar to the TRAOD algorithm, these methods cannot take full account of time, such as departure time, arrival time and duration. Chen et al. [26] proposed iBOAT (Isolation-Based Online Anomalous Trajectory) algorithm, which adds the sub-trajectory outliers recognition and online processing functions to the iBAT algorithm, without taking into account the spatio-temporal correlation properties of trajectories.

The second class focuses on the time attribute of trajectories. For example, Li et al. [27] proposed a temporal outlier detect method, emphasizing the historical similarity trend between data points. Bu et al. [20] presented a novel framework for monitoring distance-based anomalies over continuous trajectory streams. They constructed local clusters based on the local continuity feature of trajectory, and monitored anomalies by effective pruning strategies. This method concentrates on typical scenarios of context-aware applications. Ge et al. [28] provided a top-k evolutionary trajectory outlier detection method, named as TOP-EYE, which continuously calculates the outlier score of each trajectory in a cumulative manner based on the evolutionary direction and density of trajectories. They also introduced a decay function to reduce the influence of past trajectories on the evolved outlier score. The method realizes the timely recognition of single trajectory deviation. TOP-EYE can recognize evolved outliers at a very early stage with a relatively low false alarm rate. Shen et al. [8] proposed an approach that detects outliers from vehicle trajectories to discover roaming events, in which the temporal and speed patterns are used for eliciting drivers' intentions.

Zhu et al. [21] proposed a time-dependent trajectory outlier detection algorithm TPRO. Considering that the detect results of TRAOD algorithm may be influenced by irrelevant trajectories, TPRO compares each trajectory with its associated conventional routes to find out all the outliers in historical trajectory dataset. But it cannot detect new trajectories in real time. To this end, the same research team upgraded the TPRO algorithm in the literature [29] and proposed a real-time trajectory outlier detection algorithm TPRRO based on time-dependent conventional routes, including offline preprocessing and real-time detection steps.

Most existing trajectory outlier detection methods only consider the time or location characteristics of trajectories, ignoring the integration of several specific characteristics including velocity, direction and spatio-temporal correlation. In addition, the research aiming at the above problem shown in Fig. 1b is rare. Therefore, further research and application on trajectory outlier detection are urgently needed.

3 Preliminary concept and problem definition

Definition 1 (*Time-stamped location* [30]) Let t be a timestamp and (x, y) be a location in $\mathbb{R}^2$. A time-stamped location is defined as a triple (t, x, y), which means that an object is at location (x, y) at time t.

Hereinafter, triple and location will be used as synonyms for time-stamped location.

Definition 2 (*Trajectory*) A trajectory is composed of an identification and an ordered set of time-stamped locations, denoted as T.

$$T = \{Tid, (t_1, x_1, y_1), (t_2, x_2, y_2), \ldots, (t_n, x_n, y_n)\}, \quad (1)$$

where $t_r < t_{r+1}$ for all $1 \leq r < n$, Tid is the identification of a trajectory, and n is the number of sampled points in the trajectory.

A dataset of p trajectories is represented as $TS = \{T_1, T_2, \ldots, T_p\}$, and the size of TS is denoted as $|TS|$ ($|TS| = p$).

Definition 3 (*Length of trajectory*) Consider a trajectory T. The length of trajectory T is defined as the number of triples in T. It is denoted as $|T|$.

For example, $|T| = n$ in (1).

Definition 4 (*Sub-trajectory* [30]) Consider a trajectory S. $S = \{Sid, (t_{i_1}, x_{i_1}, y_{i_1}), (t_{i_2}, x_{i_2}, y_{i_2}), \ldots, (t_{i_m}, x_{i_m}, y_{i_m})\}$ is defined as a sub-trajectory of T in (1), denoted as $S \preccurlyeq T$, where $1 \leq i_1 < \cdots < i_m \leq n$.

Definition 5 (*Trajectory segment*) Consider the i-th trajectory T_i in TS. A trajectory segment is defined as the line segment between any pair of adjacent time-stamped locations in T_i. We denote the q-th segment of T_i as $TSeg_q^i$.

$$TSeg_q^i = (loc_q^i loc_{q+1}^i)(1 \leq q < n), \quad (2)$$

where n is the number of triples in T_i, loc_q^i and loc_{q+1}^i are respectively the q-th and (q+1)-th locations in T_i.

Accordingly, $T_i = \{TSeg_q^i | q = 1, \ldots, n-1\}$. And a set of all the trajectory segments in T_i, denoted as $TSegs^i$, is represented as follows:

$$TSegs^i = \{(loc_q^i loc_{q+1}^i) | q = 1, \ldots, n-1\}. \quad (3)$$

As is shown in Fig. 2, $n = 8$, T_i contains 7 trajectory segments $(loc_1^i loc_2^i), \ldots, (loc_7^i loc_8^i)$, $TSeg_4^i$ is annotated.

Definition 6 (*Trajectory slice*) Consider the i-th trajectory T_i in TS. A trajectory slice is defined as a sequence of several adjacent trajectory segments in T_i. We denote the r-th slice of T_i as $TSlice_r^i$.

$$TSlice_r^i = (loc_{p_r}^i \ldots loc_{p_{r+1}}^i)(r = 1, \ldots, z), \quad (4)$$

where z is the number of trajectory slices in T_i. Accordingly, $T_i = \{TSlice_r^i | r = 1, \ldots, z\}$. And a set of all the trajectory slices in T_i, denoted as $TSlices^i$, is represented as follows:

$$TSlices^i = \{(loc_{p_1}^i \ldots loc_{p_2}^i), (loc_{p_2}^i \ldots loc_{p_3}^i), \ldots, (loc_{p_z}^i \ldots loc_{p_{z+1}}^i)\}, \quad (5)$$

where $1 = p_1 < p_2 < \cdots < p_z < p_{z+1} = n$.

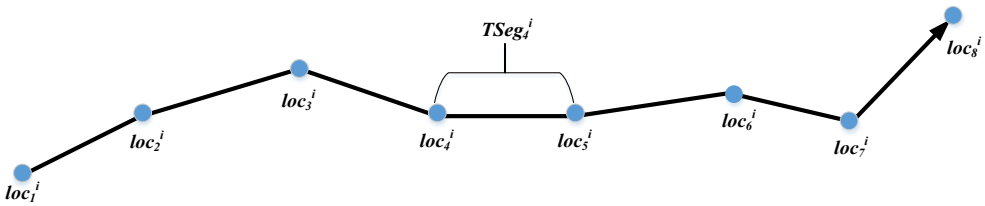


Fig. 2 Schematic diagram of trajectory segment

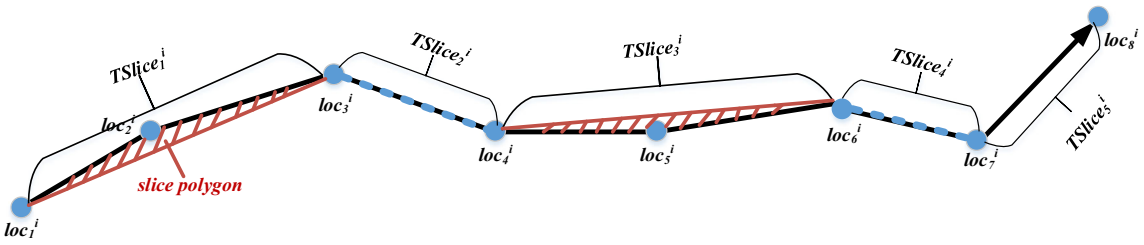


Fig. 3 Schematic diagram of trajectory slice

As is shown in Fig. 3, $n = 8$, T_i contains 5 trajectory slices. For example, $(loc_1^i loc_2^i loc_3^i)$ is the first trajectory slice $TSlice_1^i$.

Definition 7 (*Stay point*) A stay point, denoted as loc_{sp}, refers to the location where the dwell time exceeds a certain threshold. It can be defined as a four tuple $(spid, x, y, \Delta t)$, where $spid$, (x, y) and Δt respectively represent the identifier, coordinate and duration of the stay point.

In general, a stay point is not a single point, it represents the place where people move around. For example, a person is shopping in the same mall for a long time, the positions where he passes through are within a certain range.

Definition 8 (*Slice area*) Slice area refers to the area of the polygon consisting of multiple trajectory segments connected end to end in the same trajectory slice. As is shown in Fig. 3, each shadow region is referred to as a slice polygon.

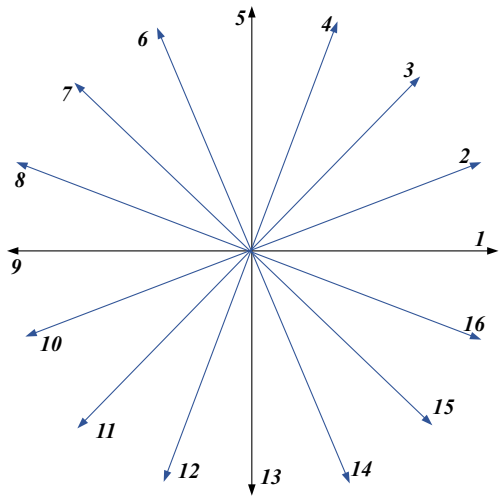


Fig. 4 The 16 direction regions with continuous integer codes

4 Generation of trajectory slices

4.1 Trajectory direction-code sequence

Direction is one of the important properties of trajectory. Our method extracts the direction feature of each trajectory segment in order to represent the moving trend of a trajectory. Firstly, we evenly divide the 2D-plane coordinate system into N regions according to the angle. A specific direction code is assigned to each region, and the angle of each region is $2{\cdot}\pi/N$. Let N be 16, Fig. 4 shows N direction regions with continuous integer codes, ranging from 1 to 16. Hereinafter, integer code is also known as direction code (or direction-code).

The slope of a directed line segment can reflect its direction. Therefore, for each trajectory T_i (assuming its length is n), we calculate the slope k_q of each trajectory segment $TSeg_q^i (1 \le q < n)$. The calculation equation is as follows:

$$k_q = \frac{y_{q+1}^i - y_q^i}{x_{q+1}^i - x_q^i}. \tag{6}$$

Based on the value of k_q, we classify each trajectory segment into a definite direction region with the corresponding direction code. It is worth noting that if the value falls on the boundary of two direction regions, we take the bigger code. If x_{q+1}^i is equal to x_q^i, there are three cases: (1) If y_{q+1}^i is equal to y_q^i, the direction code of $TSeg_q^i$ is assigned as 0, representing the locations are identical at two continuous timestamps t_q and t_{q+1}. (2) If y_{q+1}^i is bigger than y_q^i, the direction code of $TSeg_q^i$ is assigned as 5. (3) If y_{q+1}^i is smaller than y_q^i, the direction code of $TSeg_q^i$ is assigned as 13. Consequently, a dataset of trajectory direction-code sequences, denoted as t_dcs, is obtained.

4.2 Trajectory inflection points and trajectory slices

In this paper, we detect trajectory outliers based on the partitioned slices. In detail, we divide each trajectory into

Table 1 Notations used in IPG algorithm

Notation	Description
$tralen$	The number of trajectories
D or t_dcs_i	The direction-code sequence of the i-th trajectory
$Dlen$	The number of segments in the i-th trajectory
out	The inflection points set of the i-th trajectory
D_x	The x-th direction code in D

several slices by analyzing the sequence of trajectory direction codes to find inflection points where the trajectory directions change abruptly. The segment directions within each slice are highly similar with each other. The inflection points of each trajectory are calculated by Algorithm 1, denoted with IPG. The notations used in IPG algorithm are listed in Table 1.

Algorithm 1 IPG: Inflection Points Generation

Input: t_dcs(a dataset of direction-code sequences), θ_c(direction-code threshold), N(the number of direction codes)

Output: $t\_inflections$(a dataset of inflection-point sequences)

1: $tralen \leftarrow$ length(t_dcs);
2: **for** $i \leftarrow 1$ to $tralen$ **do**
3: $D \leftarrow t\_dcs_i$;
4: $Dlen \leftarrow$ length(D);
5: $out \leftarrow \{1\}$; //initialization, the index of the first point;
6: $flag \leftarrow 0$;
7: **for** $p \leftarrow 1$ to $Dlen$-1 **do**
8: **if** $flag == 0$ **then**
9: $kb \leftarrow p$;
10: **end if**
11: $ke \leftarrow p$+1;
12: **if** $D_{ke} == 0$ **then**
13: $flag \leftarrow 1$;
14: continue;
15: **end if**
16: $flag \leftarrow 0$;
17: **if** (abs($D_{kb} - D_{ke}$) $>= \theta_c$) and (abs($D_{kb} - D_{ke}$) $<= N - \theta_c$) **then**
18: $out \leftarrow out \cup \{ke\}$;
19: **end if**
20: **end for**
21: $out \leftarrow out \cup \{Dlen + 1\}$;
22: $t\_inflections \leftarrow t\_inflections \cup out$;
23: **end for**
24: **return** $t\_inflections$;

Let $Dlen_{max}$ be the maximum value of all $Dlen$ values, $Dlen_{avg}$ be the average value of all $Dlen$ values. The time complexity of Algorithm 1 is $O(tralen \cdot Dlen_{avg})$. The space complexity of Algorithm 1 is $O(tralen \cdot Dlen_{max})$, which is mainly due to storing the direction-code sequences of all the trajectories.

In this paper, N=16. Lines 8 - 16 of Algorithm 1 are used to address the problem that direction code is 0. Furtherly, based on Algorithm 1, we optimize the partitioning result by comparing each slice area with the area threshold θ_a. For each trajectory, if there is a slice area being bigger than θ_a, the value of θ_c will be reduced by one for each iteration until θ_c is reduced to 0. Consequently, the updated $t\_inflections$ will be generated.

In the following algorithms, the optimal $t\_inflections$ set is adopted. On the basis of $t\_inflections$, each trajectory can be partitioned into several slices. Each slice consists of several trajectory segments. Therefore, each slice has a corresponding set of direction codes, which will be integrated into a unified code later.

4.3 The distance between homogeneous trajectory slices

The distance calculation between two slices is the main tool for trajectory outlier detection. We group trajectory slices into several clusters according to the direction features of slices. And our distance calculation method is thus proposed based on slice clusters. The distance between homogeneous trajectory slices implies potential trajectory direction features, so the direction factor is not needed to be emphasized in distance calculation equation. Here, the homogeneous trajectory slices refer to the slices with same direction code. Therefore, our distance calculation model is provided as follows.

The components of distance equation are intuitively illustrated in Fig. 5, where $TSlice_r^i$ is the r-th slice in the i-th trajectory, $TSlice_s^j$ is the s-th slice in the j-th trajectory, $TSeg_q^i$ is the q-th segment in the i-th trajectory, $TSeg_t^j$ is the t-th segment in the j-th trajectory, $Slicelen_r^i$ is the path length of $TSlice_r^i$, $Slicelen_s^j$ is the path length of $TSlice_s^j$, $Slicedt_{loc}$ is the distance between the central points of two slices, just as O^i and O^j shown in the figure. For any slice, the location information of its central point is calculated based on the average latitude and longitude of all locations in this slice.

As is shown in Fig. 5, the distance between two homogeneous slices is calculated as follows:

$$Slicedt = w_{loc} \cdot Slicedt_{loc} + w_{plen} \cdot Slicedt_{plen}, \quad (7)$$

where $Slicedt$ represents the distance between two slices, $Slicedt_{loc}$ and $Slicedt_{plen}$ respectively represent the location

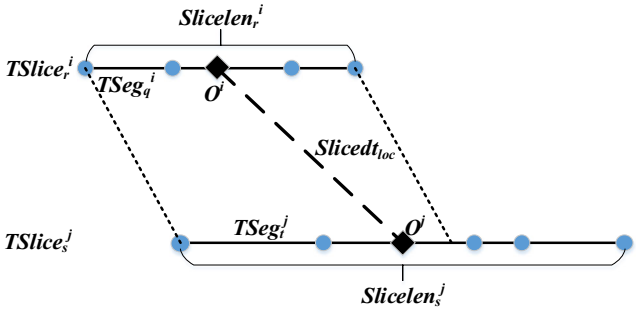


Fig. 5 The distance between two homogeneous slices with same direction code

distance and path distance, w_{loc} and w_{plen} respectively represent the weight for $Slicedt_{loc}$ and the weight for $Slicedt_{plen}$. In our experiment, $w_{loc} = w_{plen} = 0.5$. The calculation method of $Slicedt_{plen}$ is:

$$Slicedt_{plen} = |\frac{Slicelen_s^j}{\Delta t_s^j} - \frac{Slicelen_r^i}{\Delta t_r^i}|, \tag{8}$$

where Δt_r^i represents the time interval of $TSlice_r^i$, and Δt_s^j represents the time interval of $TSlice_s^j$.

Based on Algorithm 1, the data acquisition of each slice for each trajectory is implemented by Algorithm 2, denoted with DATS. The notations used in DATS algorithm are listed in Table 2.

Table 2 Notations used in DATS algorithm

Notation	Description
$t\_inflections_i$	The inflection-point sequence of the i-th trajectory
$inflsnum$	The number of inflection points in the i-th trajectory
dc_b	The index of the first sampling point within a slice
dc_e	The index of the last sampling point within a slice
$TSlices\_dc_j^i$	The direction code sequence of the j-th slice in the i-th trajectory
$spdcs_j^i$	The union direction code of the j-th slice in the i-th trajectory
$t\_sliceavg_j^i$	The average information of the j-th slice in the i-th trajectory
$interval$	The time interval within one trajectory slice
$avglat$	The average latitude within one trajectory slice
$avglong$	The average longitude within one trajectory slice
$TSlices\_dst_j^i$	The path length of the j-th slice in the i-th trajectory

Algorithm 2 DATS: Data Acquisition of Trajectory Slices

Input: t_dcs(a dataset of direction-code sequences), $t\_inflections$(a dataset of inflection-point sequences)

Output: $spdcs$(a dataset of union direction-code sequences of slices for each trajectory), $t\_sliceavg$(a dataset of the average information of slices for each trajectory), $TSlices\_dst$(a dataset of the path-length sequences of slices for each trajectory)

1: $tralen \leftarrow$ length(t_dcs);
2: **for** $i \leftarrow 1$ to $tralen$ **do**
3: $\quad infls \leftarrow t\_inflections_i$;
4: $\quad D \leftarrow t\_dcs_i$;
5: $\quad inflsnum \leftarrow$ length($infls$);
6: $\quad$ **for** $j \leftarrow 1$ to $inflsnum$-1 **do**
7: $\quad\quad dc_b \leftarrow infls_j$;
8: $\quad\quad z \leftarrow j + 1$;
9: $\quad\quad dc_e \leftarrow infls_z - 1$;
10: $\quad\quad TSlices\_dc_j^i \leftarrow D(dc_b : dc_e)$; //The direction code sub-sequence extracted from D, with subscript from dc_b to dc_e
11: $\quad\quad spdcs_j^i \leftarrow$ round(mean($TSlices\_dc_j^i$));
12: $\quad\quad t\_sliceavg_j^i \leftarrow [interval, avglat, avglong]$;
13: $\quad\quad TSlices\_dst_j^i \leftarrow \Sigma_{q=dc_b}^{dc_e-dc_b}$ distance($TSeg_q^i$);
14: $\quad$ **end for**
15: **end for**
16: **return** $spdcs$, $t\_sliceavg$, $TSlices\_dst$;

Let $inflsnum_{max}$ be the maximum value of all $inflsnum$ values, $inflsnum_{avg}$ be the average value of all $inflsnum$ values. The time complexity of Algorithm 2 is $O(tralen{\cdot}inflsnum_{avg})$. The space complexity of Algorithm 2 is $O(tralen{\cdot}inflsnum_{max})$, which is mainly due to storing the inflection-point sequences of all the trajectories.

5 Trajectory outlier detection based on common slices sub-sequence

5.1 Common slices sub-sequence

To address the problem represented in Fig. 1, we firstly find the common slices sub-sequences (abbr. CSS for single one, and CSSes for mulitple ones) of two trajectories. As is shown in Fig. 6, T_i and T_j are two trajectories in TS, the sub-sequences $\{TSlice_3^i, TSlice_4^i, TSlice_5^i, TSlice_6^i\}$ and $\{TSlice_4^j, TSlice_5^j, TSlice_6^j, TSlice_7^j\}$ are a pair of CSSes between T_i and T_j. Because both of them have four corresponding trajectory slices with identical direction-code. Based on CSSes, we calculate the distance between any two trajectories.

The common slices sub-sequences for each trajectory are obtained by Algorithm 3, denoted with GCSS.

Algorithm 3 GCSS: Get Common Slices Sub-sequences

Input: $tralen$, $spdcs$
Output: $tcell$(a dataset of the corresponding subscripts information of all pairs of CSSes between two trajectories)

1: **for** $i \leftarrow 1$ to $tralen$ **do**
2: **for** $j \leftarrow 1$ to $tralen$ **do**
3: **if** $j == i$ **then**
4: continue;
5: **end if**
6: $tx \leftarrow spdcs_i$; //a sequence of union slice direction-codes in the i-th trajectory
7: $ty \leftarrow spdcs_j$; //a sequence of union slice direction-codes in the j-th trajectory
8: **if** isempty(tx) or isempty(ty) **then**
9: continue;
10: **end if**
11: $[lcslen, txb, txe, tyb, tye] \leftarrow \text{LCS}(tx, ty)$;
12: $t \leftarrow [lcslen, txb, txe, tyb, tye]$;
13: $len \leftarrow \text{length}(ty)$;
14: $prety \leftarrow ty$;
15: **while** $tye + 1 < len$ **do**
16: $pretye \leftarrow tye$;
17: $ty \leftarrow prety(tye + 1 : len)$; //The sub-sequence extracted from $prety$, with subscript from $tye + 1$ to len
18: $[lcslen, txb, txe, tyb, tye] \leftarrow \text{LCS}(tx, ty)$;
19: $tyb \leftarrow tyb + pretye$;
20: $tye \leftarrow tye + pretye$;
21: **if** $txb == 0$ or $txe == 0$ **then**
22: break;
23: **end if**
24: $t \leftarrow t \cup \{[lcslen, txb, txe, tyb, tye]\}$;
25: **end while**
26: $tcell_{i,j} \leftarrow t$; //the dataset of CSSes between the i-th and j-th trajectories
27: **end for**
28: **end for**
29: **return** $tcell$;

In Lines 11 and 18 of Algorithm 3, the function LCS calculates the longest common sub-sequence between two character sequences. As described, each trajectory has been transformed to a direction-code sequence. All the direction-codes are ordinary characters, therefore, the LCS function is suitable to solve our trajectory problem. The parameters $lcslen, txb, txe, tyb, tye$ respectively represent the length of the longest common sub-sequences (noted as lcs), the index of the first common slice in tx, the index of the last common slice in tx, index of the first common slice in ty, index of the last common slice in ty.

Algorithm 3 calculates the CSSes for each trajectory, so the time complexity of Algorithm 3 depends on the number of trajectories and the length of the union direction-code sequence of each trajectory. Specifically, let $spdcslen_{max}$ be the maximum length of the dataset $spdcs$, the time complexity of LCS function is $O(spdcslen_{max}^2)$. For each trajectory, it is needed to scan all the other trajectories to find the CSSes. The length of the union direction-code sequence of each trajectory is far less than the number of trajectories, that is, $spdcslen_{max} \ll tralen$. For each pair of trajectories, Lines 15 - 25 is used to obtain the other CSSes except for the longest one. The execution time for this part can be ignored because the remain length of $spdcs_i$ is very small. Therefore, the overall time complexity of Algorithm 3 is $O(tralen^2 \cdot spdcslen_{max}^2)$. The space complexity of Algorithm 3 is $O(tralen^2)$, which is mainly due to storing the dataset $tcell$.

5.2 Trajectory outlier detection

Considering the continuous features of trajectories, we propose a trajectory outlier detection algorithm based on common slices sub-sequence. The schematic diagram is provided in Fig. 7. The above work introduced in Sections 4 and 5.1 is the basis of detection.

5.2.1 The distance between two common slices sub-sequences

The distance between two common slices sub-sequences, noted as *CSSdt*, is calculated based on the average value of all the distances between each pair of corresponding slices in two CSSes. In any pair of CSSes, the direction codes of each pair of slices are correspondingly equal. Therefore, according to Section 4.3, the calculation method of distance between such pair of homogeneous slices is based on (7)–(8).

Definition 9 (*CSS distance*) CSS distance refers to the distance between two common slices sub-sequences. Without loss of generality, consider a pair of corresponding CSSes $CSS^i = \{TSlice^i_{b_i}, \ldots, TSlice^i_{e_i}\}$ and $CSS^j = \{TSlice^j_{b_j}, \ldots, TSlice^j_{e_j}\}$. The CSS distance between CSS^i and CSS^j, denoted as *CSSdt*, is calculated as follows:

$$CSSdt = \frac{1}{e_i - b_i} \sum_{k=1}^{e_i - b_i} Slicedt_k, \tag{9}$$

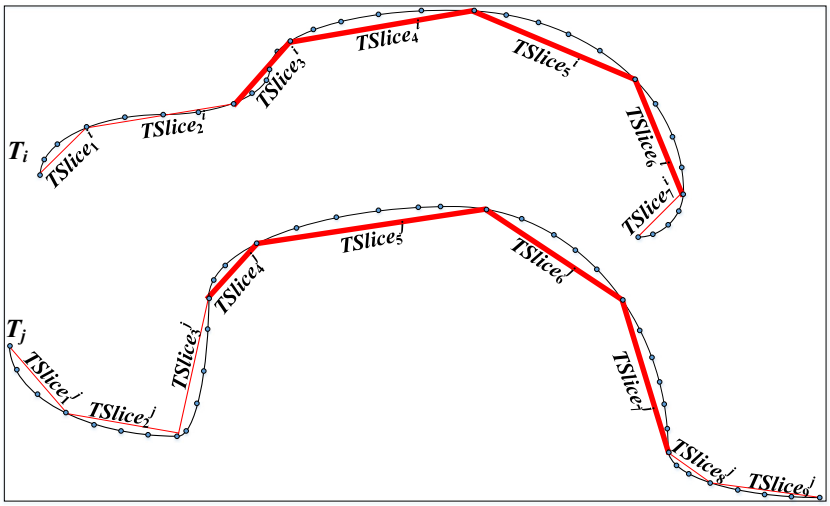


Fig. 6 A pair of common slices sub-sequences between trajectories T_i and T_j

where CSS^i and CSS^j represent a pair of CSSes between the i-th and the j-th trajectories, $Slicedt_k$ represents the distance between the k-th pair of homogenous slices, it is calculated using (7)–(8). $TSlice^i_{b_i}$ represents the b_i-th slice in the i-th trajectory, $TSlice^i_{e_i}$ represents the e_i-th slice in the i-th trajectory, being similar to $TSlice^j_{b_j}$ and $TSlice^j_{e_j}$. b_i and e_i respectively represent the first and last subscripts in CSS^i, being similar to b_j and e_j. In addition, the value of $e_i - b_i$ is equal to the one of $e_j - b_j$. The value represents the number of slices in CSS^i or CSS^j.

It is noted that the number of CSSes between two trajectories is not necessarily only one. We find all the CSSes from any pair of trajectories, and then calculate the min CSS distance to evaluate the distance between two trajectories.

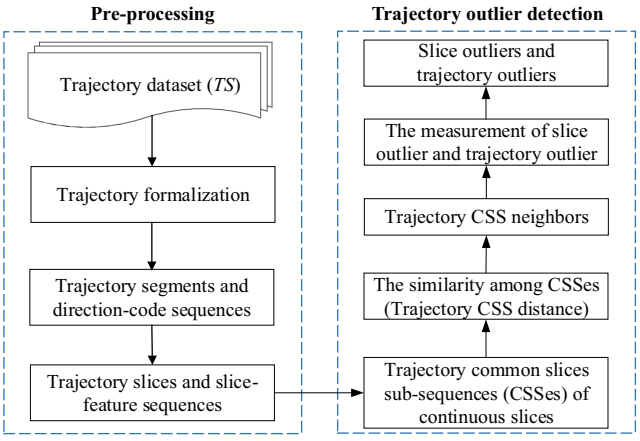


Fig. 7 Schematic diagram for the proposed trajectory outlier detection method

Definition 10 (*CSS neighbors*) Consider a trajectory T in TS. Its CSS neighbors refer to the other trajectories in TS which have common slices sub-sequences with T.

5.2.2 Trajectory outlier detection based on CSS

Definition 11 (*Slice outlier*) A trajectory slice is called a slice outlier (or an outlying slice) if the number of its neighbors is less than the designated threshold, where the neighbors refer to the other trajectory slices with a small distance from it.

Definition 12 (*Trajectory outlier*) $\forall T_i \in TS$, it is called as a trajectory outlier (also known as an outlying trajectory) if the number of its neighbors is less than designated threshold, where the neighbors refer to the other trajectoris in TS with a small distance from T_i.

In our method, a trajectory is identified as an outlying one if one of the following conditions is met:

1) if the trajectory has no CSSes with any other trajectories in TS,
2) if the number of its CSS neighbors is less than the threshold θ_{cnt},
3) if the ratio of its $cssn$ to $|TS|$ is less than θ_{per}, where $cssn$ is the number of CSS neighbors with CSS distance being greater than θ_{cssd}.

Our trajectory outlier detection algorithm based on CSS is presented in Algorithm 4, denoted with TODCSS. The notations used in TODCSS algorithm are listed in Table 3.

Table 3 Notations used in TODCSS algorithm

Notation	Description
$tcss$	The dataset of the corresponding subscripts information of a pair of CSSes between two trajectories
$slice\_sub$	The subscript information of any slice in a pair of CSSes
t_cssd	The dataset of all the distances between any two corresponding slices in a pair of CSSes
$cssdij$	The average value of t_cssd
t_cssdij	The dataset of several $cssdij$ values
$cssdcell$	The dataset of all the CSS distances between any pair of trajectories
$cssdm$	The dataset of the min CSS distances between any pair of trajectories

Algorithm 4 TODCSS: Trajectory Outlier Detection based on CSS

Input: $tcell$, θ_{cssd}(CSS distance threshold),θ_{cnt}(number threshold of CSS neighbors),θ_{per}(ratio threshold of CSS neighbors)
Output: $traoutlier$(a dataset of trajectory outliers)

1: $cssdcell \leftarrow \phi$;
2: **for** each $tcss$ in $tcell$ **do**
3: $t\_cssdij \leftarrow \phi$;
4: **for** each $slice\_sub$ in $tcss$ **do**
5: $t\_cssd \leftarrow$ the distances between each pair of slices based on $slice\_sub$ //using (7)–(8)
6: $cssdij \leftarrow$ mean(t_cssd);
7: $t\_cssdij \leftarrow t\_cssdij \cup \{cssdij\}$;
8: **end for**
9: $cssdcell \leftarrow cssdcell \cup \{t\_cssdij\}$;
10: **end for**
11: $cssdm \leftarrow$ min($cssdcell$);
12: $t\_cnt \leftarrow \phi$;
13: **for** each trajectory in $cssdm$ **do**
14: $t\_cnt \leftarrow t\_cnt \cup$ {the number of its CSS neighbors};
15: **end for**
16: $traoutlier \leftarrow$ {find($t\_cnt == 0$)}$\cup$ {find($t\_cnt < \theta_{cnt}$)};
17: $matr \leftarrow$ the rows of $cssdm$;
18: **for** $i \leftarrow 1$ to $matr$ **do**
19: $cc \leftarrow$ {find($cssdmat(i,:) > \theta_{cssd}$)};
20: $per \leftarrow$ length(cc)$/matr$;
21: **if** $per > \theta_{per}$ **then**
22: $traoutlier \leftarrow traoutlier \cup \{i\}$;
23: **end if**
24: **end for**
25: **return** $traoutlier$;

The time complexity of Algorithm 4 depends on the following three aspects:

(a) Lines 2 - 10 : the time for calculating $cssdcell$, whose time complexity is $O(n_{css} \cdot tralen^2)$, where n_{css} is the number of CSSes between two trajectories, $tralen$ is the number of trajectories, $n_{css} \ll tranlen$. For Dataset1 used in later experiments, the maximum value of n_{css} is 10, it is far less than $tralen$ (855). Therefore, the approximate time complexity of this part is $O(tralen^2)$. (b) Lines 13 - 15 : the time for calculating the vector t_cnt based on $cssdm$, the time complexity is $O(tralen)$. (c) Lines 18 - 24 : it is used to find trajectory outliers, whose time complexity is $O(matr)$, where $matr$ is equal to $tralen$.

The time complexity of the other lines is $O(1)$. So the total approximate time complexity of Algorithm 4 is $O(tralen^2) + O(tralen) + O(tralen) + O(1)$. In conclusion, its overall time complexity is $O(tralen^2)$.

6 Experiments

We perform a set of experiments to evaluate the accuracy and efficiency of the proposed algorithm.

The experiments are conducted with Matlab 8.3 (64-bit) on a PC with Intel (R) Core (TM) 2 Duo CPU 2.6 GHz and 8 GB of RAM. The operating system is Windows 7. Three datasets (Section 6.1) are used. Some experimental results of TODCSS are compared with the prior work of TRAOD [1]. The parameters allocation is presented in Section 6.2.

According to Algorithms 1- 4, we complete the trajectory outlier detection and achieve the experimental results comparison and analysis.

6.1 Dataset

6.1.1 Real hurricane trajectory dataset

We use a real trajectory dataset: the Atlantic hurricane track dataset [31]. This dataset is called Best Track, which contains the hurricane's position in latitude and longitude, maximum sustained winds in knots, and central pressure in millibars at 6-hourly (0000, 0600, 1200, 1800 UTC) intervals. We extract the time, latitude and longitude from Best Track for experiments. We use the Atlantic hurricanes from the years 1851 through 2013. This dataset has 855 trajectories and 30146 points, denoted as Dataset1. A small portion (years 1990-2013) of Dataset1, which has 152 trajectories

and 6557 points, is also used. The smaller dataset is denoted as Dataset2.

6.1.2 Real-life mobility trajectory dataset

We also use a real-life dataset of taxi moving trajectories [32, 33]. This dataset is denoted as RtTS. It contains GPS coordinates of approximately 500 taxis collected in the San Francisco Bay Area during May 2008. The average time interval between two consecutive locations is less than 10 seconds [33]. The format of each mobility trace file is as follows: each line contains latitude, longitude, occupancy and time, where the occupancy is ignored in our experiments. Since the trajectory of a taxi during an entire month can hardly be considered a single trajectory, we use the method in the literature [30] to pre-process this dataset. Specifically, the trajectory data of the day between May 25 at 12:04 and May 26 at 12:04 is extracted because there was the highest concentration of locations in the dataset during this period. After completing the trajectory filtering and position interpolation, we obtained 480 trajectories and 244 locations per trajectory on average.

6.1.3 Synthetic labeled dataset

In order to evaluate the accuracy of trajectory outlier detection, a trajectory dataset with reference classes is required. Therefore, we generate a dataset of synthetic labeled trajectories using the publicly available trajectory generator program written by Piciarelli.[1]

A set of 1000 normal trajectories from 10 different trajectory clusters and another set of 30 abnormal trajectories from 30 different clusters are created with the randomness parameter set to the default value 0.7, which is set referring to Literature [7]. After being added time dimension, the dataset contains 1030 random three-dimensional trajectories of length 20. This dataset, denoted as SynTS, is used as one reference dataset in our experiments.

6.2 Parameter determining

The values we used for θ_c, θ_a, θ_d, and θ_{cssd} are clearly dependent on dataset and application.

We set initial θ_c as 3, representing the angle difference between two consequent trajectory segment is from $\pi/4$ to $\pi/2$. This interval is enough to reflect the turning of direction. Then, θ_c will be updated according to the slice areas in each trajectory. If the designated condition that any slice area is larger than θ_a is met, the value of θ_c will be reduced by one for each iteration until θ_c is reduced to 0.

The value of θ_a is calculated based on the first partition result. For all of the slices in which all the direction codes are identical, we calculate their slice areas in order to obtain a set of such areas. Compared to the other slices, the direction change of the trajectory segments is minimal in such slices, and accordingly, the slice areas are also minimal. Therefore, choosing this area set as a basis for the area threshold setting can achieve a good effect of trajectory partition. We set θ_a as the approximate value at the 96th percentile in the sorted set of slice areas.

Being similar to the above method, the distance threshold θ_d is set as the approximate value at the 95th percentile in the sorted set of distances between each pair of slices, in which all the direction codes are identical. The CSS distance threshold θ_{cssd} is set as the approximate value at the 90th percentile in the sorted set of CSS distances between each pair of CSSes.

Although the parameters in our experiments are determined based on the above optimal principle, in order to represent the different results with different parameters, we change the values of some parameters to obtain contrast results which will be provided in Section 6.3.

6.3 Experimental results

6.3.1 Experiments on the real hurricane trajectory dataset

1) The slices effect of trajectory partition

Owing to the large number of trajectories in each dataset, we provide the partition results of the first 10 trajectories in each dataset in order to clearly show the effect of the proposed method. It is shown in Fig. 8, where Fig. 8a and b respectively show the running results on the first 10 trajectories of Dataset1 and Dataset2. We use different color and linewidth to distinguish the adjacent trajectory slices.

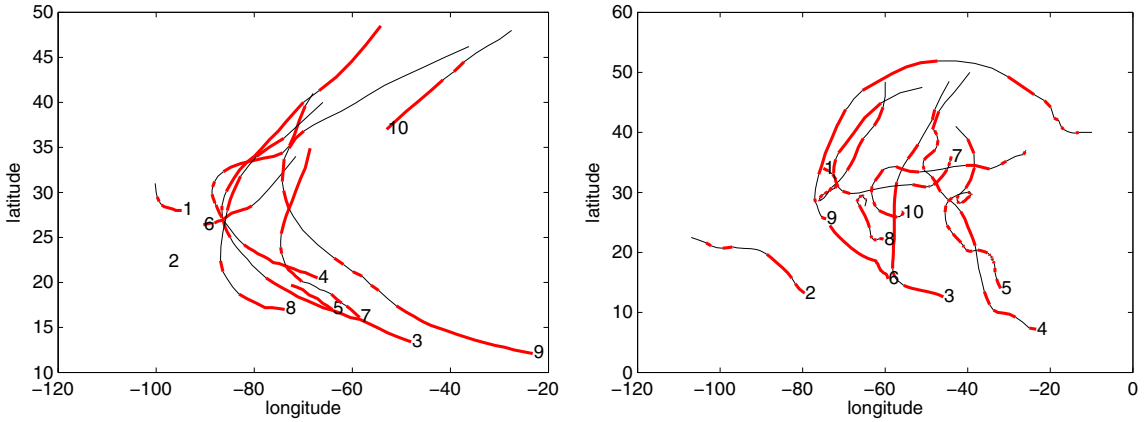
One can observe from Fig. 8 that the effect of our trajectory partition method is excellent. There are significant changes in direction at the inflection points. The direction codes within each slice are highly similar with each other.

2) Detection results of slice outliers

In this section, slice outliers are detected based on the abnormal ratios of slices. If its abnormal ratio is larger than the parameter θ_r, the slice will be detected as an outlying slice. Here, θ_r refers to the abnormal ratio threshold of slices. The value of θ_r is set as the approximate value at the 98th percentile in the sorted set of abnormal ratios. The parameter θ_d is set according to Section 6.2.

Figure 9a and b respectively show the slice outliers detected by our method based on Dataset1 and Dataset2. In Fig. 9, the slice outliers are marked in bold red lines, while the normal trajectories are marked in fine black lines.

[1] http://avires.dimi.uniud.it/papers/trclust/create_ts2.m.



(a) Dataset1 (b) Dataset2

Fig. 8 The slices representation of the first 10 trajectories in Dataset1 and Dataset2

From Fig. 9, we can see some slice outliers are obviously different with other slices in trajectory dataset. For example, the outlying slices marked in bold red lines on top-right corner in Fig. 9a, and the outlying slices marked in bold red lines on middle-right corner in Fig. 9b.

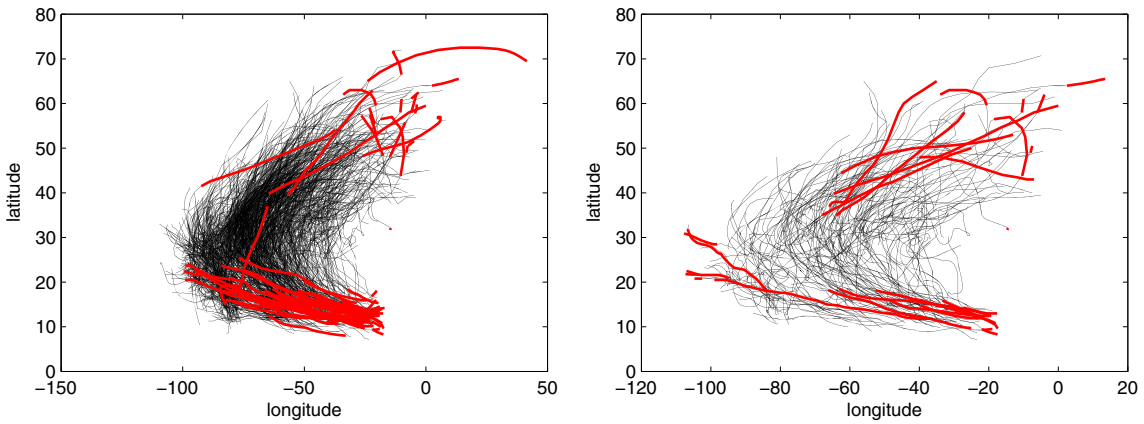
In order to illustrate the detection results with different values of parameter θ_r, we vary it from 99th to 90th percentile in the sorted set of ratios, which are calculated based on the number of abnormal slices to the number of homogeneous slices. Tables 4 and 5 show the numbers of slice outliers and the ratios of $|SO|$ to $|SS|$ with different θ_r. In Table 4, SO refers to the set of slice outliers detected from Dataset1, SS refers to the set of all the slices in Dateset1. In Table 5, SO refers to the set of slice outliers detected from Dataset2, SS refers to the set of all the slices in Dateset2. Here, $|SO|$ and $|SS|$ respectively represent the number of elements in SO and SS.

From Tables 4–5, it can be seen that the number of slice outliers is getting bigger when the value of θ_r is becoming smaller. Because θ_r is the distance threshold, when it is becoming smaller, the number of distances being bigger than θ_r will be bigger. If the value of θ_r is the 95th percentile or larger than it, the ratio of $|SO|$ to $|SS|$ is less than 5%, which is the general percentage criteria of outlier detection [34]. According to the specific application, we can select an appropriate value of the parameter. The running results provided here can serve as a reference.

3) Detection results of trajectory outliers

In this section, trajectory outliers are detected using Algorithm 4, where θ_{per} is set as a value in the range of 0.3∼0.4, θ_{cnt} is set as $0.05{\cdot}|TS|$, and the other parameters are set according to Section 6.2. Here, $|TS|$ represents the number of trajectories in TS.

Figure 10a and b respectively show the trajectory outliers detection results based on Dataset1 and Dataset2. In Fig. 10a, each blue isolated point represents a trajectory with only a single sampling point. These trajectories are obviously the outliers, which can be detected using our method. In addition, trajectories marked in bold red lines are



(a) Dataset1 (b) Dataset2

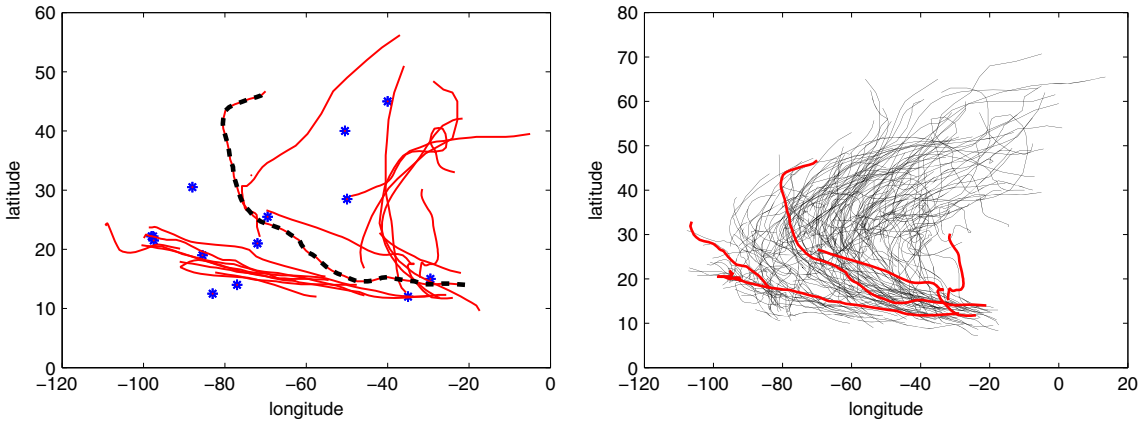
Fig. 9 Slice outliers detected from Dataset1 and Dataset2

Table 4 The number of slice outliers for Dataset1 with different values of θ_r

θ_r	99th 0.4951	98th 0.4125	97th 0.3427	96th 0.2968	95th 0.2562	90th percentile 0.1767
The number of slice outliers ($\|SO\|$)	95	191	286	382	477	954
The ratio of $\|SO\|$ to $\|SS\|$	0.93%	1.87%	2.81%	3.75%	4.68%	9.36%

Table 5 The number of slice outliers for Dataset2 with different values of θ_r

θ_r	99th 0.425	98th 0.3676	97th 0.3333	96th 0.3049	95th 0.2853	90th percentile 0.2216
The number of slice outliers ($\|SO\|$)	19	38	58	77	96	192
The ratio of $\|SO\|$ to $\|SS\|$	0.73%	1.45%	2.21%	2.94%	3.67%	7.33%



(a) Dataset1 (b) Dataset2

Fig. 10 Trajectory outliers detected from Dataset1 and Dataset2

Table 6 The number of trajectory outliers for Dataset1 with different values of θ_{cssd}

θ_{cssd}	99th 28.3	98th 25.08	97th 23.31	96th 22.07	95th 21.06	90th percentile 17.79
The number of trajectory outliers ($\|TO\|$)	30	31	32	33	33	47
The ratio of $\|TO\|$ to $\|TS\|$	3.51%	3.63%	3.74%	3.86%	3.86%	5.5%

also the outlying ones which have several sampling points within them. Because the number of trajectories in Dataset1 is large, the normal trajectories are not depicted in Fig. 10a in order to represent the effect of isolated points. In Fig. 10b, the trajectory outliers are marked in bold red lines, and the normal trajectories are marked in fine black lines.

As is shown in Fig. 10a, the trajectory marked in black dotted line is the one hit Cape Fear in North Carolina on September 1996, which is a large destructive hurricane [13]. The visual inspection results show that TODCSS algorithm effectively detects the trajectory outliers.

In order to illustrate the detection results with different values of parameter θ_{cssd}, we vary it from 99th to 90th percentile for Dataset1 and 98th to 84th percentile for Dataset2. Tables 6 and 7 show the numbers of trajectory outliers and the ratios of $|TO|$ to $|TS|$ with different θ_{cssd}. In Table 6, TO refers to the set of trajectory outliers detected from Dataset1, TS refers to the set of all the trajectories in Dateset1. In Table 7, TO refers to the set of trajectory outliers detected from Dataset2, TS refers to the set of all the trajectories in Dateset2. Here, $|TO|$ represents the number of elements in TO.

From Tables 6 and 7, it can be seen that the number of trajectory outliers is getting bigger when the value of θ_{cssd} is becoming smaller. Because θ_{cssd} is the CSS distance threshold, when it is becoming smaller, the number of CSS distances being bigger than θ_{cssd} will be bigger. For Dataset1, if the value of θ_{cssd} is the 95th percentile or larger than it, the ratio of $|TO|$ to $|TS|$ is less than 5%. For Dataset2, if the value of θ_{cssd} is the 90th percentile or larger than it, the ratio of $|TO|$ to $|TS|$ is less than 5%. An appropriate value of the parameter can be selected based on the specific application.

6.3.2 Experiments on the real-life mobility trajectory dataset

1) The slices effect of trajectory partition

Owing to the large number of trajectories and the large length of each trajectory in RtTS, we provide the partition results of the first two trajectories in this dataset in order to clearly show the effect of the proposed method. It is shown in Fig. 11. Similarly, we use different color and linewidth to distinguish the adjacent trajectory slices.

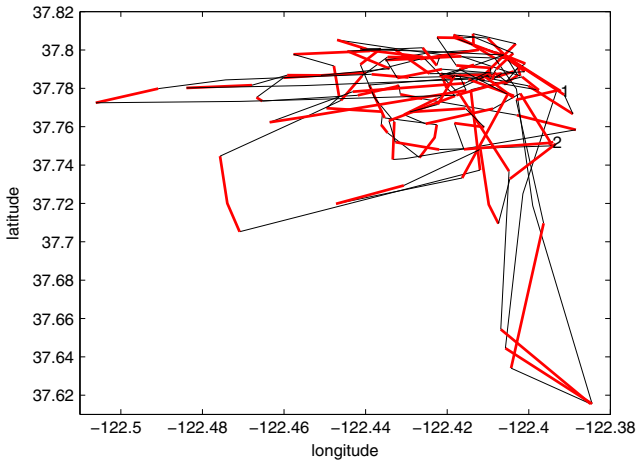


Fig. 11 The slices representation of the first two trajectories in RtTS

As can be seen from Fig. 11, it is clear that the effect of our trajectory partition method is excellent. There are significant changes in direction at the inflection points. The direction codes within each slice are highly similar with each other.

2) Detection results of slice outliers

In this section, slice outliers are detected based on the abnormal ratios of slices. If its abnormal ratio is larger than the parameter θ_r, the slice will be detected as an outlying slice. Here, θ_r refers to the abnormal ratio threshold of slices. The parameter θ_d is set according to Section 6.2.

In order to illustrate the detection results with different values of parameter θ_r, we vary it from 99th to 90th percentile in the sorted set of ratios, which are calculated based on the number of abnormal slices to the number of homogeneous slices. Table 8 shows the numbers of slice outliers and the ratios of $|SO|$ to $|SS|$ with different θ_r. Here, SO refers to the set of slice outliers detected from RtTS, SS refers to the set of all the slices in RtTS. $|SO|$ and $|SS|$ respectively represent the number of elements in SO and SS.

It can be seen from Table 8 that the number of slice outliers is getting bigger when the value of θ_r is becoming smaller. The same reason has been provided in Section 6.3.1. If the value of θ_r is the 95th percentile or larger than it, the ratio of $|SO|$ to $|SS|$ is less than 5%. According to the specific application, we can select

Table 7 The number of trajectory outliers for Dataset2 with different values of θ_{cssd}

θ_{cssd}	98th 25.76	96th 22.97	94th 20.73	92th 19.37	90th 18.19	88th 17.20	86th 16.42	84th percentile 15.72
The number of trajectory outliers ($\|TO\|$)	1	1	2	5	6	8	11	13
The ratio of $\|TO\|$ to $\|TS\|$	0.66%	0.66%	1.32%	3.29%	3.95%	5.26%	7.24%	8.55%

Table 8 The number of slice outliers for RtTS with different values of θ_r

θ_r	99th	98th	97th	96th	95th	90th percentile
	0.289	0.1942	0.0955	0.0491	0.0341	0.0232
The number of slice outliers ($\|SO\|$)	704	1416	2125	2834	3542	7071
The ratio of $\|SO\|$ to $\|SS\|$	0.99%	2.00%	3.00%	4.00%	5.00%	9.98%

an appropriate value of the parameter. The running results provided here can serve as a reference.

3) Detection results of trajectory outliers

In this section, trajectory outliers are detected using Algorithm 4, where θ_{per} is set as 0.35, θ_{cnt} is set as $0.05{\cdot}|TS|$, and the other parameters are set according to Section 6.2.

Figure 12 shows the detection result of trajectory outliers based on RtTS, where the trajectory outliers are marked in bold red lines, and the normal trajectories are marked in fine black lines.

As can be seen from Fig. 12, some trajectory outliers are obviously different with other trajectories in RtTS. For example, the outlying trajectories marked in bold red lines on top-left corner and on top-right corner.

In order to illustrate the detection results with different values of parameter θ_{cssd}, we vary it from 98th to 84th percentile for RtTS. Table 9 shows the numbers of trajectory outliers and the ratios of $|TO|$ to $|TS|$ with different θ_{cssd}. Here, TO refers to the set of trajectory outliers detected from RtTS, TS refers to the set of all the trajectories in RtTS. $|TO|$ represents the number of elements in TO.

It can be seen from Table 9 that the number of trajectory outliers is getting bigger when the value of θ_{cssd} is becoming smaller. The same reason has been provided in Section 6.3.1. If the value of θ_{cssd} is larger than the 86th percentile in the sorted set of CSS distances, the ratio of $|TO|$ to $|TS|$ is less than 5%. An appropriate value of the parameter can be selected based on the specific application.

The common results of the experiments on Dataset1, Dataset2 and RtTS can provide a valuable reference for parameter determination in future research.

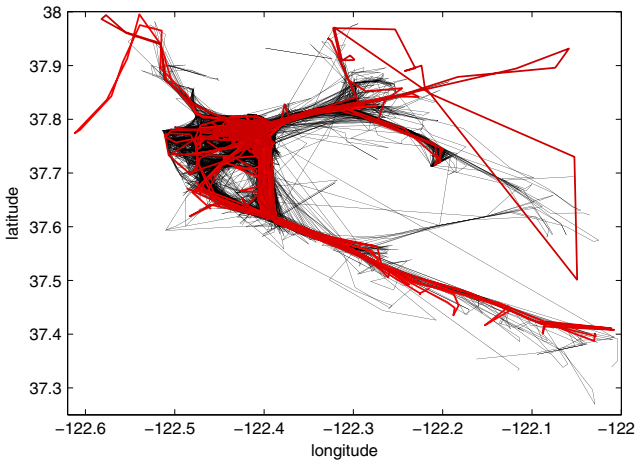


Fig. 12 Trajectory outliers detected from RtTS

6.3.3 Experiments on the synthetic labeled dataset

In this section, we conduct a set of experiments to compare our proposed algorithm TODCSS with the algorithm TRAOD [1], which is one of the most popular trajectory outlier detection algorithms based on trajectory partition.

In order to evaluate the accuracy of trajectory outlier detection, as described in Section 6.1, we generate the dataset SynTS using the publicly available trajectory generator program. It is shown in Fig. 13. The 3-dimensional effects of these synthetic trajectories are shown in Fig. 13a, while their projections in the 2D-plane are shown in Fig. 13b. Note that for clarity, different trajectory clusters are represented with different colors in Fig. 13. The trajectories marked with black solid lines are labeled abnormal.

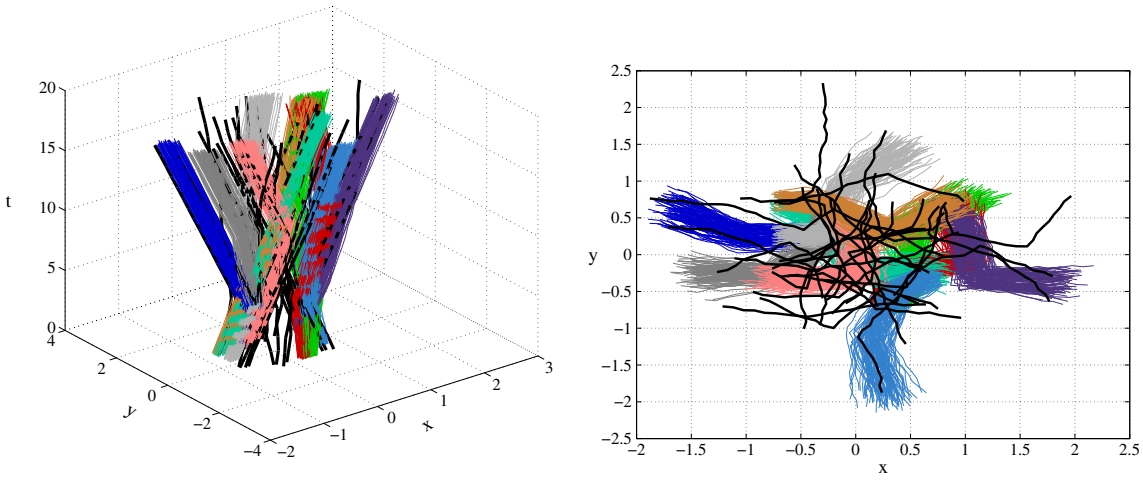
1) The slices effect of trajectory partition

As described in Section 6.3.1, owing to the large number of trajectories in this dataset, we provide the partition results of the trajectories being labeled abnormal. The proposed algorithm is compared with TRAOD. The slices effect of trajectory partition is shown in Fig. 14, where Fig. 14a and b respectively show the running results of TODCSS and TRAOD on the trajectories being labeled abnormal. We use different color and linewidth to distinguish the adjacent trajectory slices.

From the intuitive results shown in Fig. 14a, there are significant changes in direction at the inflection points. The direction codes within each slice are highly similar with each other. From the intuitive results shown in Fig. 14b, each trajectory is evenly partitioned and some t-partitions contain obvious inflection points. For example, there are obvious direction changes in the slices in the middle and lower region and in the middle right region. This means that some of the original trajectory trend information may be lost in trajectory partition using TRAOD algorithm. It is

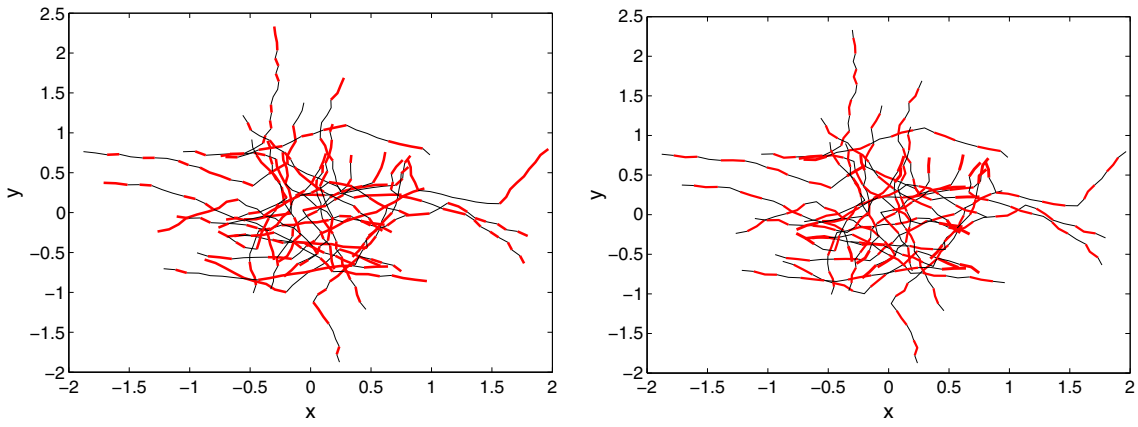
Table 9 The number of trajectory outliers for RtTS with different values of θ_{cssd}

θ_{cssd}	98th 0.0223	96th 0.0171	94th 0.0148	92th 0.0134	90th 0.0124	88th 0.0116	86th 0.011	84th percentile 0.0104
The number of trajectory outliers ($\|TO\|$)	3	4	11	16	19	21	24	26
The ratio of $\|TO\|$ to $\|TS\|$	0.63%	0.83%	2.29%	3.33%	3.96%	4.38%	5.00%	5.42%



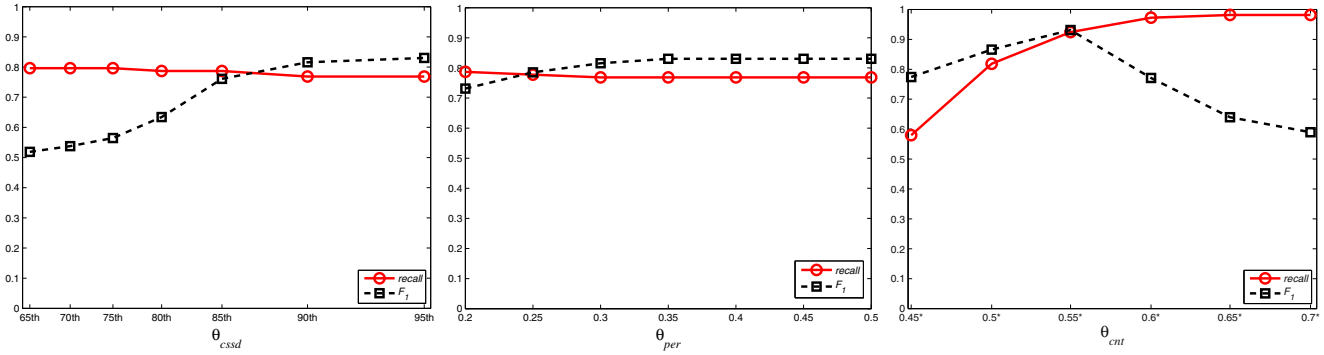
(a) in the 3D-plane (b) in the 2D-plane

Fig. 13 The synthetic labeled dataset SynTS



(a) TODCSS (b) TRAOD

Fig. 14 The slices representation of the trajectories being labeled abnormal of algorithms TODCSS and TRAOD (Note that for clarity, only 30 abnormal trajectories in the dataset SynTS are represented. The concept of slices in TODCSS corresponds to that of the t-partitions in TRAOD)



(a) Specified θ_{per} and θ_{cnt} (b) Specified θ_{cssd} and θ_{cnt} (c) Specified θ_{cssd} and θ_{per}

Fig. 15 F-measure and *recall* values under different parameters of TODCSS algorithm

clear that our trajectory partition method outperforms that of TRAOD.

Trajectory partition is the first step of both TODCSS and TRAOD, therefore, the partition results play an important role in trajectory outlier detection.

2) Accuracy of trajectory outlier detection

Because the reference categories have been already labeled by SynTS dataset, we firstly use F-measure to evaluate the performance of trajectory outlier detection algorithms. The F-measure (also known as the F-fraction) is associated with precision and recall for information retrieval. They are defined as follows [10]:

$$precision = \frac{TP}{TP + FP} \tag{10}$$

$$recall = \frac{TP}{TP + FN} \tag{11}$$

where TP (True Positive) represents the number of outlying trajectories being truly detected, TN (True Negative) represents the number of normal trajectories being truly detected, FP (False Positive) represents the number of normal trajectories being falsely detected as outliers, FN (False Negative) represents the number of outlying trajectories being falsely detected as normal ones. The F-measure, denoted as F_1, is computed as follows:

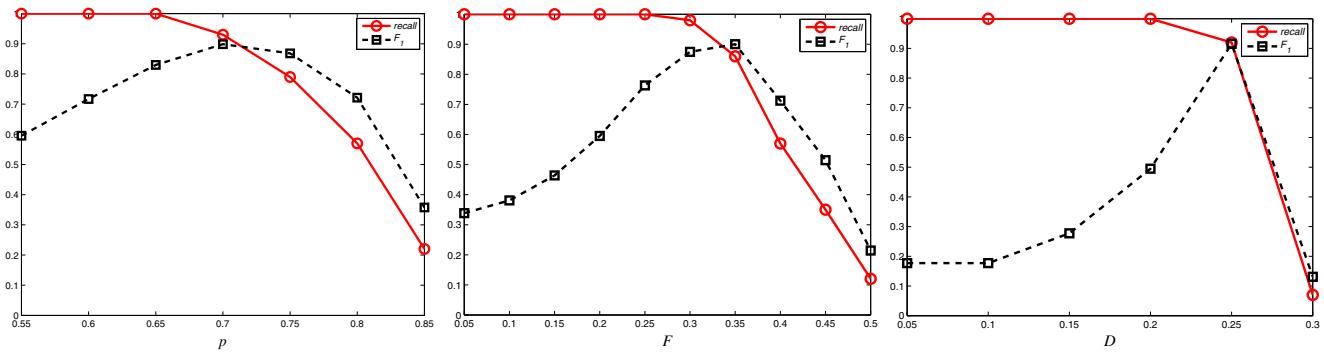
$$F_1 = \frac{2 \times precision \times recall}{precision + recall} \tag{12}$$

The range of the F-measure values is [0,1]. A higher value means that the algorithm has better performance.

A high detection rate of outliers and low false positive rate of normal points is expected from the outlier detection algorithms. However, some conflict will exist between the two criteria. A higher detection rate usually leads to a high rate of false-positives. In many applications, capturing as many outliers as possible (i.e., the sensitivity or recall of outlier detection) is more important than falsely detecting the normal objects as outliers [10]. The trajectory outlier detection algorithm is no exception. Therefore, the *recall* value is also adopted as an evaluation standard in this subsection. It is calculated based on (11).

The results of TODCSS and TRAOD on the synthetic labeled dataset are shown in Figs. 15 and 16.

(1) TODCSS algorithm



(a) Specified F and D (b) Specified p and D (c) Specified p and F

Fig. 16 F-measure and *recall* values under different parameters of TRAOD algorithm

In TODCSS algorithm, there are three main parameters: θ_{cssd}, θ_{per} and θ_{cnt}. The initial settings are: θ_{cssd} is set as described in Section 6.2, $\theta_{per} = 0.3$, θ_{cnt} is set as $0.5 \cdot |TS|$. To better verify the performance of the proposed algorithm, we respectively vary each of them to investigate the changes of F-measure and *recall* values. The other parameters used in this algorithm are set according to Section 6.2. The running results are shown in Fig. 15.

Figure 15a–c respectively represent the following three cases: a) Various θ_{cssd} with specified θ_{per} and θ_{cnt}. We vary θ_{cssd} from 65th to 95th percentile in the sorted set of CSS distances between each pair of CSSes. b) Various θ_{per} with specified θ_{cssd} and θ_{cnt}. We vary θ_{per} from 0.2 to 0.5. c) Various θ_{cnt} with specified θ_{cssd} and θ_{per}. We vary θ_{cnt} from 0.45 to 0.7 times $|TS|$. As to θ_{cssd}, θ_{per} and θ_{cnt}, when a parameter changes, the other two specified parameters keep the initial values.

Note that for clarity, only ordinals are represented on the horizontal axis in Fig. 15a. For example, the value on the location of X-axis marked "65th" refers to the 65th percentile in the sorted set of CSS distances between each pair of CSSes. In Fig. 15c, similarly, the value on the location of X-axis marked "0.45*" refers to $0.45 \cdot |TS|$.

According to the results shown in Fig. 15, we adjust the parameters in TODCSS algorithm to get the optimal results. Specifically, in order to balance the F-measure and *recall* values, we can respectively set θ_{cssd} as 95th percentile in the sorted set of CSS distances between each pair of CSSes, $\theta_{per} = 0.35$, θ_{cnt} is set as $0.55 \cdot |TS|$ for each corresponding case. The optimal settings are not far away from the initial settings, which indicates our algorithm is relatively stable.

(2) TRAOD algorithm

In TRAOD algorithm, there are three main parameters: p, F and D. The initial settings are $p = 0.55$, $F = 0.2$, $D = 0.21$. To better verify the performance of this algorithm, we respectively vary each of them to investigate the F-measure and *recall* values. The other parameters except p, F and D used in this algorithm are set according to the settings proposed in Literature [1]. The running results are shown in Fig. 16.

Figure 16a–c respectively represent the following three cases: a) Various p with specified F and D. Assuming $p \in [0.55 : 0.85]$, F is assigned 0.2, and D is assigned 0.21. b) Various F with specified p and D. Assuming $F \in [0.05 : 0.5]$, p is assigned 0.55, and D is assigned 0.21. c) Various D with specified p and F. Assuming $D \in [0.05 : 0.3]$, p is assigned 0.55, and F is assigned 0.2.

According to the results shown in Fig. 16, we adjust the parameters in TRAOD algorithm to get the optimal results. Specifically, in order to balance the F-measure and *recall* values, we can respectively set $p = 0.7$, $F = 0.3$ and $D = 0.25$ for each corresponding case.

As can be seen from Figs. 15–16, the F-measure (F_1) and *recall* values of TODCSS algorithm are superior to that of TRAOD algorithm. For one hand, the best value of F-measure (F_1) is 0.9312 for TODCSS algorithm, and that is 0.9154 for TRAOD algorithm. On the other hand, compared to TRAOD algorithm, the curves of TODCSS algorithm are more stable than that of TRAOD algorithm. That is to say, the proposed TODCSS algorithm is less dependent on the parameters. In summary, our proposed algorithm achieves the accuracy and stability in trajectory outlier detection.

7 Conclusion

In this paper, we propose a novel trajectory outlier detection algorithm TODCSS based on common slices sub-sequence. For each trajectory, the direction-code sequence is firstly calculated based on the direction of each segment. The corresponding sequences consisting of trajectory slices are furtherly obtained. And then, we measure the distance between two trajectories based on the common slices sub-sequences between them. Finally, the slice outliers and trajectory outliers are detected based on the new distance calculation. The proposed algorithm TODCSS uses the characteristics of orientation, location and continuity of each trajectory to achieve more accurate distance between two trajectories. Experimental results on real hurricane trajectory dataset, real-life mobility trajectory dataset and synthetic labeled dataset show that the proposed approach is suitable for trajectory partition and trajectory outlier detection, and the detection rate of the approach is superior to that of existing method. The proposed algorithm can be used in many geographical research fields including but not limited to bad weather forecast, intelligent transportation and detection of users' abnormal behaviors. In future researching, we plan to integrate the features of stay points of each trajectory into our research in order to reflect the reality more accurately. Because stay points are closer to points of interest (known as POI [35]), the research based on stay points is more meaningful.

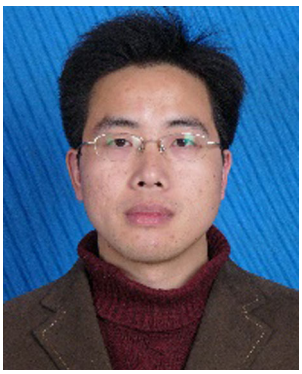
Acknowledgements The authors would like to thank the reviewers for their useful comments and suggestions for this paper. This work was supported by the National Natural Science Foundation of China (61702010, 61672039), the Key Program for University Top Talents of Anhui Province (gxbjZD2016011), the Natural Science Foundation of Anhui Province (1508085QF134), the University Natural Science Research Program of Anhui Province (KJ2017A327), and the Science and Technology Project of Wuhu City (2016cxy04).

References

1. Lee JG, Han J, Li X (2008) Trajectory outlier detection: a partition-and-detect framework. In: Proceedings of the 24th IEEE International Conference on Data Engineering (ICDE), pp 140–149
2. Su H, Zheng K, Huang J et al (2015) Calibrating trajectory data for spatio-temporal similarity analysis. VLDB J 24(1):93–116
3. Sanchez I, Aye ZMM, Rubinstein BIP et al (2016) Fast trajectory clustering using hashing methods. In: Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN), pp 3689–3696
4. Anagnostopoulos C, Hadjiefthymiades S (2014) Intelligent trajectory classification for improved movement prediction. IEEE Trans Syst Man Cybern Syst 44(10):1301–1314
5. Chen CM, Pi DC, Fang ZR (2012) Artificial immune algorithm applied to short-term prediction for mobile object location. Electron Lett 48(17):1061–1062
6. Gupta M, Gao J, Aggarwal CC (2014) Outlier detection for temporal data: a survey. IEEE Trans Knowl Data Eng 25(1):1–20
7. Laxhammar R, Falkman G (2014) Online learning and sequential anomaly detection in trajectories. IEEE Trans Pattern Anal Mach Intell 36(6):1158–1173
8. Shen M, Liu DR, Shann SH (2015) Outlier detection from vehicle trajectories to discover roaming events. Inform Sci 294:242–254
9. Laxhammar R, Falkman G (2015) Inductive conformal anomaly detection for sequential detection of anomalous sub-trajectories. Ann Math Artif Intell 74(1-2):67–94
10. Han J, Kamber M, Pei J (2013) Data mining: concepts and techniques. Morgan Kaufmann, San Francisco
11. Gan G, Ng MK-P (2017) K-means clustering with outlier removal. Pattern Recogn Lett 90:8–14
12. Albanese A, Pal SK (2014) Rough sets, kernel set, and spatiotemporal outlier detection. IEEE Trans Knowl Data Eng 26(1):194–207
13. Aggarwal CC (2017) Outlier analysis, 2nd edn. Springer International Publishing
14. Li Z, Ding B, Han J et al (2010) Swarm: mining relaxed temporal moving object clusters. In: Proceedings of the VLDB Endowment vol 3, no 1, pp 723–734
15. Ge Y, Xiong H, Liu C et al (2012) A taxi driving fraud detection system. In: Proceedings of the IEEE International conference on data mining (ICDM), pp 181–190
16. Yu Y, Cao L, Rundensteiner EA et al (2014) Detecting moving object outliers in massive-scale trajectory streams. In: Proceedings of the ACM SIGKDD international conference on knowledge discovery and data mining, vol 8, pp 422–431
17. Chen J, Abbady S, Duggimpudi MB (2016) Spatiotemporal outlier detection: did buoys tell where the hurricanes were? Papers Appl Geograph 2(3):298–314
18. Knorr EM, Ng RT, Tucakov V (2000) Distance-based outliers: algorithms and applications. The VLDB J 8(3-4):237–253
19. Li X (2007) ROAM: rule- and motif-based anomaly detection in massive moving object data sets. In: Proceedings of the SIAM international conference on data mining, pp 273–284
20. Bu Y, Chen L, Fu AW-C et al (2009) Efficient anomaly monitoring over moving object trajectory streams. In: Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining (KDD 09), pp 159–168
21. Zhu J, Jiang W, Liu A et al (2015) Time-dependent popular routes based trajectory outlier detection. In: Proceedings of the international conference on web information systems engineering, pp 16–30
22. Guan B, Zhang Y, Liu L et al (2012) An improving algorithm of trajectory outliers detection. Advances in intelligent and soft computing. Springer, Berlin, pp 907–914
23. Masciari E (2011) Trajectory outlier detection using an analytical approach. In: Proceedings of the 23rd IEEE international conference on tools with artificial intelligence, pp 377–384
24. Zhang D, Li N, Zhou Z-H et al (2011) iBAT: detecting anomalous taxi trajectories from GPS traces. In: Proceedings of the 13th ACM international conference on Ubiquitous Computing (UbiComp 11), pp 99–108
25. Mohamad I, Ali MAM, Ismail M (2011) Abnormal driving detection using real time global positioning system data. In: Proceedings of the IEEE international conference on space science and communication, pp 1–6
26. Chen C, Zhang D, Castro PS et al (2013) iBOAT: isolation-based online anomalous trajectory detection. IEEE Trans Intell Transp Syst 14(2):806–818
27. Li X, Li Z, Han J et al (2009) Temporal outlier detection in vehicle traffic data. In: Proceedings of the IEEE International Conference on Data Engineering (ICDE), pp 1319–1322
28. Ge Y, Xiong H, Zhou Z et al (2010) TOP-EYE: top-k evolving trajectory outlier detection. In: Proceedings of the 19th ACM International conference on information and knowledge management, pp 1733–1736
29. Zhu J, Jiang W, Liu A et al (2017) Effective and efficient trajectory outlier detection based on time-dependent popular route. World Wide Web 20(1):111–134
30. Domingo-Ferrer J, Trujillo-Rasua R (2012) Microaggregation- and permutation-based anonymization of movement data. Inform Sci 208:55–80
31. UNISYS (2015) Atlantic tropical storm tracking by year[EB/OL]. http://weather.unisys.com/hurricane/atlantic/
32. Piorkowski M, Sarafijanovic-Djukic N, Grossglauser M (2009) CRAWDAD dataset epfl/mobility(v 2009-02-24)[EB/OL]. https://doi.org/10.15783/C7J010
33. Piorkowski M, Sarafijanovic-Djukic N, Grossglauser M (2009) A parsimonious model of mobile partitioned networks with clustering. In: Proceedings of the 1st international conference on communication systems and NETworks, pp 1–10
34. Chen Y, Miao D, Zhang H (2010) Neighborhood outlier detection. Expert Syst Appl 37(12):8745–8749
35. Lv M, Chen L, Xu Z et al (2016) The discovery of personally semantic places based on trajectory data mining. Neurocomputing 173:1142–1153

Qingying Yu received her B.S. degree and M.S. degree from Department of Computer Science and Technology, Anhui University, Hefei, China, respectively in 2002 and 2005. Currently, she is a Ph.D. Candidate at Anhui Normal University, Wuhu, China. Her main research interests are spatial data processing and information security.

Yonglong Luo received his Ph.D. degree from the School of Computer Science and Technology, University of Science and Technology of China in 2005. Since 2007, he has been a professor in School of Mathematics and Computer Science, Anhui Normal University. Currently, he is a Ph.D. supervisor of Anhui Normal University. He is the Director of Anhui Provincial Key Laboratory of Network and Information Security. His main research interests are information security and spatial data processing.



Chuanming Chen received his B.S. degree and M.S. degree from Department of Computer Science and Technology, Anhui University, Hefei, China, respectively in 2002 and 2005. Currently, he is an associate professor at Anhui Normal University. His main research interests are data mining and intelligent computing.

Xiaohan Wang received her B.S. degree and M.S. degree from Department of Computer Science and Technology of Anhui Normal University, and Anhui University in 2001 and 2007 respectively. Currently, she is an associate professor at Anhui Normal University. Her main research interests are information security and intelligent computing.