Outlier Trajectory Detection: A Trajectory Analytics Based Approach

Zhongjian Lv[1], Jiajie Xu[1(✉)], Pengpeng Zhao[1], Guanfeng Liu[1], Lei Zhao[1], and Xiaofang Zhou[2,1]

[1] School of Computer Science and Technology, Soochow University, Suzhou, China
20165227001@stu.suda.edu.cn, {xujj,ppzhao,gfliu,zhaol}@suda.edu.cn
[2] School of ITEE, The University of Queensland, Brisbane, Australia
zxf@itee.uq.edu.au

Abstract. Trajectories obtained from GPS-enabled devices give us great opportunities to mine out hidden knowledge about the urban mobility, traffic dynamics and human behaviors. In this paper, we aim to understand historical trajectory data for discovering outlier trajectories of taxis. An outlier trajectory is a trajectory grossly different from others, meaning there are few or even no trajectories following a similar route in a dataset. To identify outlier trajectories, we first present a prefix tree based algorithm called PTS, which traverses the search space on-the-fly to calculate the number of trajectories following similar routes for outlier detection. Then we propose two trajectory clustering based approaches PBOTD and DBOTD to cluster trajectories and extract representative routes in different ways. Outlier detection is carried out on the representatives directly, and the accuracy can be guaranteed by some proven error bounds. The evaluation of the proposed methods on a real dataset of taxi trajectories verifies the high efficiency and accuracy of the DBOTD algorithm.

1 Introduction

With recent improvements in satellites and GPS-enabled devices, it is possible to accumulate a great amount of trajectory data which provides us much information about behaviors or certain variation. The vast deposits of information contained in such a big dataset can help us to understand animal migration pattern, urban traffic situation, human driving behavior and so on. So far a number of amusing applications are being developed, including routes planning [2,18], pattern mining [5,12], trajectory-based inference and prediction [8,11], moving together mining [7,20], etc.

In addition to these mentioned above, outlier trajectory detection is being paid increasingly attention in recent years for moving object surveillance. The pioneer studies [9,17,22] mainly focus on the trajectories in free-space which means that the objects tend to move freely and their movement are not restricted by a given road network. These approaches can achieve a good result but are

DOI: 10.1007/978-3-319-55753-3_15

not for those trajectories under road network constraints. Hence, it is necessary to study these trajectories limited by road network. More recently, increasing attentions [1,13,15,23,24] are paid to detect outlier moving object trajectories while considering the topology of the underlying road network. These studies help us to address some needs in our daily life, such as to identify if a taxi driver deliberately takes unnecessary detours to overcharge passengers who are not familiar to the area. Similarly, it is highly useful in monitoring the vehicles that carry noxious chemicals or other dangerous cargo. It is no doubt that outlier trajectory detection have become more important in the Internet era.

In effect, we hold that a precise definition with strong physical meaning is necessary for us to solve the problem and the result would be more accurate and convincing under the fine-grained definition. Unfortunately, previous methods seem to lack such a definition or the definition is not reasonable enough. For example, TPRO [24] gives an outlier score for given trajectory by comparing it with top-k popular routes. Apparently, it has a weak physical meaning because a normal trajectory other than the outliers may be different from the popular routes too. It means that some normal trajectories would be easily regarded as outliers. According to [6], an outlier is a data object that is grossly inconsistent with the remaining data. In this paper, we also hold that an outlier trajectory should be grossly different from the other trajectories. That is to say, if a taxi driver takes a little detour when encountering a traffic jam or does not choose a very popular route, the generated trajectory should not be regarded as an abnormal because there are still existing certain number of trajectories that are similar to it. So it is concluded that a trajectory can be seen as an outlier if there are almost no similar trajectories in dataset. It is our definition for outlier trajectory.

Although having the appropriate definition, detecting driving outliers is still a uphill work. There are mainly two challenges in our work as follows: on the one hand, there might be many different normal driving paths from the origin to the destination and it is difficult to describe all those trajectories covering these paths. So we need to cast about for a proper method to directly or indirectly portray those trajectories about their driving routes and quantity; on the other hand, when we do outlier trajectory detection based on our structure, the performance should be efficient and the result should be accurate. [15] obtained the number of relevant trajectories by comparing the given trajectory with each trajectory in dateset. There is no doubt that the accuracy is high but the efficiency will be in a dilemma when facing with a large dataset. So we should determine quickly and accurately whether a given trajectory is an outlier.

In this paper, we propose two methods to model our trajectories. Firstly, we develop a baseline algorithm called Prefix Tree Searching (PTS). In this algorithm, we generate a prefix tree by searching routes similar to a given test trajectory on the road network. Then we can get the proportion of those similar trajectories in all trajectories by this tree for outlier detection. It is time consuming although the method can give us an accurate result. Aiming to improve the performance, we present our second method based on clustering, this is because it can put similar trajectories together effectively. And then we can quickly judge whether a given trajectory is an outlier by comparing representative trajectories chosen from cluster results with the given one. Based on such idea, we present Prototype Based Outlier Trajectory Detection (PBOTD) based on the idea of k-medoids [16] as our first cluster-based algorithm and choose medoids as representative routes. But the method cannot achieve a high accurate because of the problem of the selection of k and local optimum. For overcoming these drawbacks, we present another algorithm called Density Based Outlier Trajectory Detection (DBOTD), where we refer to DBSCAN [4] to cluster and choose core routes as our representatives. The algorithm can quickly and accurately do outlier detection because the approach much more fit our definition. In summary, the main contributions of this paper include:

- To detect outliers more accurately, we map trajectories into bit-map and present PTS algorithm to calculate the proportion of similar quantity in total quantity.
- Taking efficiency into consideration, we provide a PBOTD and a better DBOTD algorithm to cluster trajectories and extract representative routes which can summarize those trajectories well and by using these representatives we can quickly do outlier detection.
- We demonstrate, by using various real dataset, that DBOTD can effectively and accurately do outlier detection compared with other algorithms.

2 Related Work

In this section, we introduce some related works which can be categorized into two groups. The first group focuses on analyzing or exploiting trajectories with research issues other than outlier detection. For example, Krumm et al. [8] and Liao et al. [11] predicted the destination and inferred transportation routines respectively. Jeung et al. [7] aimed at discovering a group of moving objects that have traveled together for some time. Tang et al. [20] presented a strong online mining method for the streaming trajectories. Chen et al. [2] defined popular routes which embody the common law of driving and designed an efficient search strategy on the given road network for popular routes. In [10], Lee et al. developed TRACLUS to cluster sub-trajectories for discovering the most common mode of movement in free-space. Other includes trajectory preprocessing like calibrating [19] and trajectory management as well as query [3,21].

The second group includes outlier trajectory detection. Lee et al. [9] put forward a group-and-detect framework and develop an algorithm called TRACLUS to detect outlier sub-trajectories in free-space. Pnueli et al. [17] proposed a new framework named ROAM to represent trajectories in a feature space oriented on the discrete fragments and developed a general-purpose, rule-space classifier to detect abnormal traces. Zhang et al. [23] proposed an isolation-based method called IBAT. The key idea is to randomly pick a point from the test trajectory and remove other trajectories which do not contain this point. If the test trajectory is an outlier, this process will end very soon. Zhu et al. [24] taked the travel time into consideration and proposed a novel algorithm called TPRO to discover those outlier trajectories during some periods which may be normal during other periods. Different from the previous ones' ideas, we hold that an outlier trajectory is grossly inconsistent with the trajectories in the dataset, which means that there are almost no trajectories following similar routes. Masciari et al. [15] keeps a similar definition but they do outlier detection by comparing the test one with each trajectory in the dataset which is unacceptable when the dataset is large enough. In contrast, we propose different methods including searching similar routes on the given road network and clustering trajectories which pass from a certain origin and a certain destination (od-pair).

3 Problem Definition

This part presents some definitions and gives a formal statement of the problem this paper focuses on.

Definition 1 (Road Network). A road network is modeled as a directed graph $G = (V, E)$, where each vertex $v_i \in V$ denotes an intersection or end of a road, and each edge $e_i^j \in E$ denotes a road segment from vertex v_i to v_j (the direction is $v_i \rightarrow v_j$).

Given a road network G, a route is a path that a moving object passes, and it is denoted as $r = (v_{c_1}, ..., v_{c_n})$, where $1 \leq c_i \leq |V|$. Trajectory data can be derived if we tackle how object moves on a route by sampling spatial and temporal information.

Definition 2 (Trajectory). A raw trajectory $t_{raw} = (p_1, p_2, ..., p_n)$ is a sequence of geo-tagged sampling points, where p_i denotes a geographic coordinate. After map-matching, we can derive the mapped trajectory $t_{mapped} = (v_{c_1}, v_{c_2}, ..., v_{c_m})$. For simplicity, we will drop the "mapped" qualifier and use t to replace t_{mapped}. We use $t_{i->j}$ to denote the sub-trajectory $(v_{c_i}, ..., v_{c_j})$ of t, where $1 \leq i < j \leq m$.

We use $Count(r)$ to denote how many trajectories or sub-trajectories cover the route r totally. A route may not be covered by existing trajectory, it means that the route's count is zero, namely $Count(r) = 0$.

Definition 3 (Route Distance Function). Route Distance Function $\varphi(r, r')$ is a formula which gives a difference score between routes which have the same origin and destination. It is noted that the distance function can also give score between a route and a trajectory. If we have a route $r = (v_{c_1}, ..., v_{c_n})$ and another route $r' = (v_{c'_1}, ..., v_{c'_m})$, where $c_1 = c'_1$ and $c_n = c'_m$, then their score can be denoted as

$$\varphi(r, r') = Min \begin{cases} \varphi(Rest(r), r') + add\_cost(e_{c_{n-1}}^{c_n}) \\ \varphi(r, Rest(r')) + add\_cost(e_{c'_{m-1}}^{c'_m}) \\ \varphi(Rest(r), Rest(r')) \\ \quad + \; replace\_cost(e_{c_{n-1}}^{c_n}, e_{c'_{m-1}}^{c'_m}) \end{cases} \tag{1}$$

where $Rest(r) = (v_{c_1}, ..., v_{c_{n-1}})$ is prefix of r after removing the last edge $e^{c_n}_{c_{n-1}}$, $add\_cost(e) = 1$ and $replace\_cost(e^{c_n}_{c_{n-1}}, e^{c'_m}_{c'_{m-1}}) = 0$ if $c_{n-1} = c'_{m-1}$ and $c_n = c'_m$, otherwise 2 which is as twice as big than *add_cost* because it takes both add and delete into consideration.

If $Rest(r) = (v_{c_1})$, then we have $\varphi(Rest(r), r') = \sum_{e \in r'} add\_cost(e)$. This also applies to the situation where $Rest(r')$ only contains starting vertex. If both $Rest(r')$ and $Rest(r)$ have one vertex, their difference score is 0.

Definition 4 (Similar Route). A route r is said to be an $\alpha - similar$ route with respect to another route r' if $\varphi(r, r') < \alpha$, where α is a given threshold.

Note that the route distance function in Definition 3 and similar route here are also applicable to a route and a trajectory because a trajectory is a route after map-matching.

Definition 5 (Outlier Trajectory). Given a trajectory t which passes from v_{c_1} to v_{c_m} and an outlier score threshold θ, we can calculate the outlier score

$$S(t) = \frac{\sum_{r \in SR(t)} Count(r)}{\sum_{r \in AR(t)} Count(r)}, \tag{2}$$

where $SR(t) = \{r \mid r\ is\ \alpha - similar\ to\ t\}$, denotes all $\alpha - similar$ routes respect to t and $AR(t) = \{r \mid r.c'_1 = c_1\ and\ r.c'_n = c_m\}$, denotes all routes which can go from t's starting vertex to t's ending vertex. If $S(t) < \theta$, then we say the trajectory is a $\theta - outlier$ trajectory.

Problem: Given a history trajectory set T, a trajectory t, a similar threshold α and an outlier score threshold θ, we need to judge whether the trajectory t is a $\theta - outlier$.

4 Road Network Search Based Approach

In this section, we firstly introduce our baseline algorithm in which we search similar routes about the given trajectory on the road network. We start from origin and traverse the road network graph to arrive at destination. The process will generate a prefix tree and so the algorithm is also called Prefix Tree Searching algorithm. Then we can do aggregation with the prefix tree for accurately detecting outliers.

4.1 Prefix Tree Based Search

Before introducing searching algorithm, we need to solve a problem on how we obtain the quantity of trajectories covering a route. For solving this, we can use a bitmap for each vertex. The bitmap records whether each trajectory goes through the vertex or not and we can get the number of the trajectories going through the vertex by it. Then we can get the quantity of trajectories covering a route by doing and-operation on the bitmaps of all vertices of this route.

Now we can introduce our prefix tree searching algorithm. Given a trajectory t, we start from the origin of t and adopt breadth-traversal to search on the road network until arriving at the destination of t. After this process, we would generate a structure like a prefix tree. However, the search space is huge and it is necessary for us to find out how to prune when searching. We mainly consider the following three aspects: firstly, if Best Match Distance between a partial route and given trajectory reaches or exceeds our threshold α, we can stop searching on this path; secondly, if there are no trajectories covering the path, namely the bitmap after and-operation is 0, we can stop too; thirdly, if the vertex is the same as another vertex in this path, we also prune the branch because a normal route usually does not contain same vertices.

It is easy to understand the second and third points. And for the first point, we firstly define the Best Match Distance function and give an example.

Definition 7 (Best Match Distance Function). Given a partial route $pr = (v_{c_1}, ..., v_{c_k})$ and a trajectory $t = (v_{c'_1}, ..., v_{c'_n})$, where $v_{c_1} = v_{c'_1}$, we can calculate their minimum route distance which is called Best Match Distance. And the Best Match Distance function $\psi_{bm}(pr, t)$ is defined as

$$\psi_{bm}(pr, t) = \varphi(part(pr), part(t)) + \sum\nolimits_{e \in rest(pr)} add\_cost(e) \qquad (3)$$

where $part(pr) = (v_{c_1}, ..., v_{separation})$, $part(t) = (v_{c'_1}, ..., v_{separation})$ and $rest(pr) = (v_{separation}, ..., v_{c_k})$. Here, a separation point $v_{separation}$ refers to the last coincidence vertex of the given trajectory and the partial route. Note that the rest part on the partial route may be empty and we do not need to calculate *add_cost* in this situation. If we arrive at destination, it is evident that $\psi_{bm} = \varphi$.

Example 1. Supposed that we have the complete trajectory $t = (v_1, v_3, v_6, v_7)$ and partial-route $pr = (v_1, v_2, v_4)$ in the left of Fig. 1. The separation point is v_1. So we can know that $part(t) = (v_1)$, $part(pr) = (v_1)$ and $rest(pr) = (v_1, v_2, v_4)$. Then the Best Match Distance can be calculated by Eq. 3 which equals 2.

Then we have the following lemma to prove the first point:

Lemma 1. *Given a trajectory t and a partial route pr_{+i}, where $i \geq 0$, we can get that $\psi_{bm}(pr_{+i}, t) \leq \psi_{bm}(pr_{+(i+1)}, t)$.* Here, we use $pr_+(i+1)$ to denote a partial route that we pass through another vertex based on pr_{+i}.

Proof. Supposed that $pr_{+i} = (v_{c_1}, ..., v_{c_i})$ and $pr_{+(i+1)} = (v_{c_1}, ..., v_{c_i}, v_{c_{i+1}})$, on the one hand, if $v_{c_{i+1}}$ is a vertex that does not exist in trajectory t, then $\psi_{bm}(pr_{+(i+1)}, t) = \psi_{bm}(pr_{+i}, t) + add\_cost(e_{c_i}^{c_{i+1}}) = \psi_{bm}(pr_{+i}, t) + 1 \geq \psi_{bm}(pr_{+i}, t)$; if $v_{c_{i+1}}$ exists in t, it is evident that $\psi_{bm}(pr_{+(i+1)}, t) = \varphi(pr_{+(i+1)}, part(t)) \geq \psi_{bm}(pr_{+i}, t)$ because *replace_cost* in $\varphi(pr_{+(i+1)}, part(t))$ not only contains add_cost but also takes deleting edges into consideration. So Lemma 1 can be proven.

Lemma 2. *Given a partial-route pr and a trajectory t, if $\psi_{bm}(pr, t) \geq \alpha$, then any route r covering pr must not be $\alpha - similar$ with t.*

Proof. We know that $pr = pr_{+0}$ and $\exists\ m \geq 1$, let $r = pr_{+m}$. According to Lemma 1 and our condition, we know that $\alpha \leq \psi_{bm}(pr, t) = \psi_{bm}(pr_{+0}, t) \leq \psi_{bm}(pr_{+1}, t) \leq ... \leq \psi_{bm}(pr_{+m}, t) = \varphi(r, t)$, so r is not $\alpha - similar$ with t by Definition 4.

Example 2. For brevity, we just show a subgraph of our road network in the left of Fig. 1 in which we can start from v_1 and go to v_7 and there are three trajectories being mapped into vertices' bitmaps. We can find that v_8 is another vertex that a moving object can go to from v_1. The vertex's bitmap is 000 which means that the three trajectories do not pass through this vertex.

If we want to test a trajectory $t = (v_1, v_3, v_6, v_7)$ and the similar threshold α is set to 2, we can get a prefix tree (in the right of Fig. 1) when searching on the road network. We firstly do and-operation on the bitmaps of origin v_1 and destination v_7. Then the result is taken as the bitmap of root node which ensures our search marching to the destination. Now we can start from the origin and search all routes which are similar to our given trajectory. We can find that we stop searching at v_8 because the bitmap after doing and-operation is 000. v_4 is also a stop node because we can get $\psi_{bm} = 2$ and this score has reached the preset limit. The path $(v1, v3, v6, v7)$ is a valid path and so we get a similar route about our test trajectory.

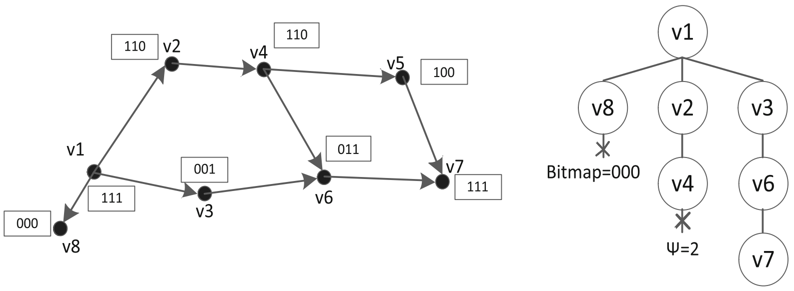


Fig. 1. A SubGraph of Road Network and A Prefix Tree of Searching

4.2 Outlier Detection

After searching, we can get paths result set and each path represents a similar route and contains a bitmap after doing and-operation. Intuitively, we can compute $Count(r)$ by bitmap. However, it has the following faults:

1. A trajectory which covers similar route's reverse vertices will also be contained in this bitmap, which is easy to understand.
2. Some trajectories will be contained in another bitmap which also results in the wrong number of $Count(r)$ as shown in Observation 1.

Observation 1. In Fig. 2, we can get two routes: $r_1 = (v_1, v_2, v_3)$ and $r_2 = (v_1, v_3)$. If we use bitmap to calculate $Count(r)$, then $Count(r_1) = 2$ and $Count(r_2) = 3$. However, there are only three trajectories which proves that our result is wrong because there are two trajectories accidentally being contained in the bitmap of r_2.

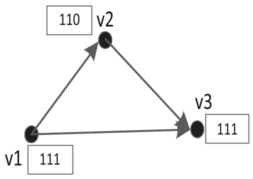


Fig. 2. An Observation of Computing Count(r)

For solving it, we need to do verification. After searching, we achieve all similar routes from prefix tree. For each route r, we can get a last bitmap resulting from doing and-operation on all bitmaps of vertices in this route and the last bitmap contains all possible trajectories. Then we compare these trajectories with this route to compute $Count(r)$. By doing so, we can do outlier detection by Eq. 2.

Though baseline algorithm can provide an accurate result, it is not efficient in two aspects: searching and verifying. So we are required to vigorously pursue other methods to describe trajectories and do outlier detection efficiently under certain precision.

5 Trajectory Clustering Based Approach

To solve the efficiency problem discussed above, this section introduces our approaches based on clustering trajectories. Clustering analysis is a division of data into groups according to given rules for similarity and it is a common means in data mining which can help us describe similar things. After putting similar trajectories into together, it is necessary to extract routes from each group that can represent clusters at a high level. Then we can do outlier detection by comparing these representatives with the test trajectory. Compared with the road network searching method, it is no doubt that the kind of method can be faster and the accuracy will not be lost too much if we choose representative routes properly.

Based on these ideas, we propose two cluster-based outlier trajectory detection algorithms: PBOTD based on prototype-based clustering and DBOTD based on density-based clustering. To make result of trajectory clustering meaningful, it is necessary to divide trajectories and sub-trajectories into different groups according to od-pairs in advance. The bitmap structure mentioned in Sect. 4.1 can help us solve the problem. For improving performance of clustering, those trajectories covering a certain route can be as one object to cluster.

We can build a prefix tree to get these routes and their quantity. Now we can cluster them and save necessary cluster's representative routes. At last, we get related routes and do outlier detection when a test trajectory comes.

5.1 Prototype Based Outlier Trajectory Detection

Prototype-based clustering algorithms, such as k-means [14], are frequently used to find the structure of a dataset by grouping all objects into clusters based on their similarities. These algorithms initialize a prototype and then update it by some strategies to achieve the final cluster results. Among them, k-means is a basic method. However, the calculation of the mean for trajectories is very difficult and it is easily affected by abnormal trajectories because of poor quality of dataset. Then we decide to refer to another algorithm called k-medoids. The algorithm divides data into k clusters and each cluster has a medoid which we can use to represent the cluster. It is a great advantage that we only save k routes for each od-pair and compare k times when doing outlier detection.

Algorithm Description. After deriving all routes subject to an od-pair, we use k-medoids method to cluster the routes and find representative routes. At first, we should select initial k medoids. By default we can use Route Distance Function to calculate distance, however, it is more appropriate to take the quantity of trajectories covering these routes into consideration. And the weighted route distance function used in this algorithm is given belows:

Definition 8 (Weighted Route Distance Function). Given two routes r and r', in order to measure their weighted distance (taking quantity into account), we define the distance function as follows:

$$\varphi_q(r, r') = \frac{\varphi(r, r')}{Count(r) + Count(r')} \tag{4}$$

where $Count(r)$ and $Count(r')$ are quantity of r and r' respectively. It is obvious that the larger the quantity is, the smaller the distance should be.

Then we can calculate an initial score for each route r_j by the following formula:

$$s_j = \sum_{i=1}^{n} \frac{\varphi_q(r_i, r_j)}{\sum_{l=1}^{n} \varphi_q(r_i, r_l)} \tag{5}$$

And we select k routes having the first k smallest scores as initial medoids, which tends to select k most middle routes. Secondly, we obtain initial cluster result by assigning each route to the nearest medoid and calculate the sum of distances from all routes to their medoids. At last, we need to choose a new route which has the minimal sum of distances from it to other routes in this cluster to update current medoid for each cluster and assign again. We will not stop these steps until the sum of distances from all routes to their medoids does not change. Note that the distance we mention here is calculated by Eq. 4.

After clustering, we need to choose routes to represent clusters which are used to detect outliers. In this algorithm, we choose the medoid as our representative route for each cluster. Then we can get a representative routes set $representative\_routes$.

Outlier Detection. Given a test trajectory t, the starting and ending vertices of t need be used to get related representative routes set $representative\_routes$. The main issue is how to make use of these routes from trajectory clustering for outlier detection. Then we need to calculate a Min Core Distance which means the minimum distance between representative routes and a trajectory. We have the following definition:

Definition 9 (Min Core Distance Function). Given a trajectory t, we can get its relevant $reference\_routes$, and the Min Core Distance Function $\rho(t)$ is defined as

$$\rho_{mc}(t) = min(\varphi(r_1, t), ..., \varphi(r_n, t)) \tag{6}$$

where $r_1, ..., r_n \in reference\_routes$.

If $\rho_{mc}(t) < \alpha$,where α is similarity threshold given, we can say that the trajectory can appear in the cluster. And a trajectory is an outlier if it does not appear in the clusters. It means that if $\rho_{mc}(t) \geq \alpha$, then the test trajectory t is an outlier. However, we cannot guarantee by theory that the result satisfies our definition and we also cannot give out the corresponding error bound.

5.2 Density Based Outlier Trajectory Detection

As mentioned above, PBOTD cannot provide guarantee of the high accuracy of the result. To address this issue, we provide another algorithm called DBOTD. In this algorithm, we refer to the idea of density-based clustering algorithms which expands result of clustering on the connectivity between the objects from the angle of their density. Correspondingly, we cluster trajectories according to the quantity of them nearby a certain route and it is extremely consistent with our outlier definition.

Algorithm Description. DBSCAN [4] is a classical algorithm in density-based cluster methods. It can discover clusters of arbitrary shape and filter out outliers. Here we apply this algorithm to cluster trajectory data. After deriving all clusters, we extract representative routes which can generalize the whole in each cluster. Now we can do outlier trajectory detection by these routes.

For portraying the compactness degree of objects, DBSCAN defines core object and connectivity on the help of neighborhood parameters $(\epsilon, MinPts)$. Similarly, we need to define a core route as follows:

Definition 10 (Core Route). Given a radius d and a MinPtsRate γ, if a route r satisfies $|N(r,d)| \geq \gamma \times |N(group\ of\ r)|$, where $N(group\ of\ r) = AR(r)$ denotes all routes which go from r's starting vertex to r's ending vertex, $N(r,d) = \{r' | \varphi(r,r') < d\ and\ r' \in N(group\ of\ r)\}$ and $|N| = \sum_{r \in N} Count(r)$, then we can say r is a core route.

According to our definitions, we should set $\gamma = \theta$ and $d = \alpha$. By doing so, we discover from Definition 10 that a core route must be a non-outlier route because the proportion of those trajectories around the core route in all trajectories reaches or exceeds our outlier threshold θ.

In this algorithm, we also expand clusters on the basis of connectivity and cores. We first choose a core route cr and then $N(cr,d)$ should also be included in this cluster. Then if there are new core routes being included in this cluster, these new core routes will be expanded too according to prior step. After these steps, we can get a certain number of clusters.

Now we need to choose representative routes to describe clusters comprehensively. On the one hand, if we record only one representative route, it is prone to generate error; on the other hand, it wastes storage space to store all core routes. For solving it, when we expand clusters from a core route, the core route will be chosen if existing routes not be clustered around it. It means that a core route will not be chosen if it can be deduced by other core routes.

Outlier Detection. After clustering, we also need to judge whether a trajectory is outlier. Namely, given a trajectory t, if Min Core Distance $\rho_{mc}(t) \geq \alpha$, it is clear that the trajectory is an outlier. We have the following lemma:

Lemma 3. *Given a trajectory t, if $\rho_{mc}(t) \geq \alpha$ wrt. $\gamma = \theta$ and $d = \alpha$, then the trajectory is an outlier.*

Proof. We know that $\rho_{mc}(t)$ denotes the minimum route distance between t and representative routes. And these routes describe clusters comprehensively. So $\rho_{mc}(t) \geq \alpha$ shows that the trajectory is not be included by any clusters. Then the trajectory must not be a core route. It means that $|N(t,d)| < \theta \times |N(group\ of\ t)|$, namely $\sum_{r \in SR(t)} Count(r) < \theta \times \sum_{r \in AR(t)} Count(r)$. So it is an outlier.

One problem an acute reader would immediately notice is that border routes in the cluster are also outliers because they also do not satisfy our definition. However, there is an error bound: we can be sure that these border routes are not-outliers within the scope of the $2d$ and we have the following lemma:

Lemma 4. *Given a trajectory t, if $\rho_{mc}(t) < \alpha$ wrt. $\gamma = \theta$ and $d = \alpha$, it must not be an outlier with the scope of the $2d$.*

Proof. From the condition $\rho_{mc}(t) < \alpha$, we know that there is at least a core route cr making $\varphi(cr,t) < \alpha$. So $N(t,2\alpha)$ must contains $N(cr,\alpha)$, where $N(t,2\alpha) = \{r' | \varphi(t,r') < 2\alpha\}$ and $N(cr,\alpha) = \{r' | \varphi(cr,r') < \alpha\}$. We know that $|N(cr,\alpha)| \geq \theta \times |N(group\ of\ cr)|$, then $|N(t,2\alpha)| \geq \theta \times |N(group\ of\ t)|$, where $N(group\ of\ cr) = N(group\ of\ t)$, which proves obviously our lemma.

Compared with the PBOTD algorithms, this algorithm can guarantee giving out a theoretic error bound. It can extract routes which represent clusters better and do outlier detection efficiently.

6 Experimental Study

This section presents our experiment of study. We introduce our experiment settings and evaluation criteria in the first subsection. Then we present experiment results and analysis in the second subsection.

6.1 Experiment Settings

The experiment is taken under a real-world dataset which contains 5,660,692 trajectories in Beijing. We picked up about 5,300 trajectories from the dataset as testing set and asked volunteers to manually label whether each trajectory is outlier or not. The rest trajectories are used as the training dataset. And the Beijing road network data in our experiment contains 253,180 vertices and 557,134 edges.

Table 1. Default values of parameters

Parameter	Default value	Description
θ	3%	Threshold of outlier score
α	25	Threshold of similar routes
k	6	K clusters in PBOTD
D	5000 K	The number of training trajectories

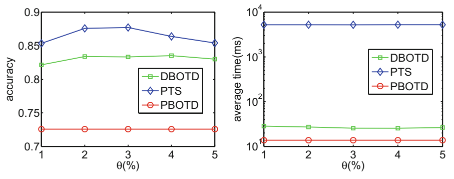
We compare the average time cost of detection and the accuracy in the next subsection. The default values for parameters are given in Table 1. In the experiments, we vary one parameter and keep the others constant to investigate the effect of this parameter. All algorithms are implemented in Java and run on a server with two 6-core Intel(R) Xeon(R) CPUs (2.6GHz) and 256GB memory.

6.2 Performance Evaluation

In this part, we vary the values of parameters in table 1 to compare our three algorithms and investigate the effect of each parameter. After this, we compare our best DBOTD algorithm with TPRO and IBAT algorithms which are proposed in [24] and [2] respectively.

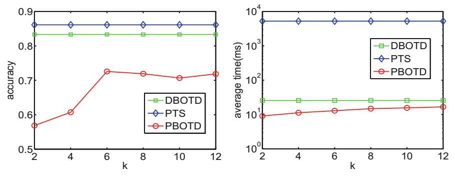
Effect of θ. In the first part of experiments, we study the effect of θ which mainly affects our PTS and DBOTD algorithm. As shown in Fig. 3(a), our PTS algorithm achieves high accuracy compared with other algorithms and DBOTD algorithm is also good which pales slightly beside PTS. Within our expectations, PBOTD works worst. In the aspect of average time costing, as shown in Fig. 3(b), PTS consumes too much time which we cannot accept while PBOTD and DBOTD can detect quickly within tens of milliseconds. Taking into account of two aspects, it holds that DBOTD works best. We can also find out that the result is stable, especially in DBOTD, when θ varies from 1% to 5%.

Effect of k. We study the effect of different k which only PBOTD has. From Fig. 4(a), we can find out that the accuracy of PBOTD is relatively poor when k is small and it will increase as k grows until reaching the limit which is not satisfactory. On the other hand, the average time of detecting a trajectory in PBOTD almost linearly increases with k increasing, which is shown in Fig. 4(b) Although PBOTD just saves k routes for each od-pair, the problem of low accuracy will make us choose other algorithms.



(a) Accuracy (b) Average time

Fig. 3. Effect of θ



(a) Accuracy (b)Average time

Fig. 4. Effect of k

Effect of α. We investigate the performance of these algorithms when the threshold of similar route α is varying. Figure 5 shows the results of our experiment. With the increase of α, PTS will incur significantly time cost while cluster-based algorithms will not because they have saved corresponding information which is used to detect outliers. On the other hand, DBOTD and PTS always outperform PBOTD in accuracy when α varies from 15 to 35. It is also shown that PTS and DBOTD have a stable and good performance in an interval of 20 and 30, which is what we expect. Taking both accuracy and time cost into consideration, DBOTD is a great algorithm.

Effect of D. In order to evaluate the effect of D, we sample the dataset to generate training dataset with different number of trajectories varying from 1000 K to 5000K, and report the average time cost and the accuracy in Fig. 6 There is a marginal increase in the regard of accuracy with the training data size growing. Meanwhile, PTS and DBOTD algorithms are always better than PBOTD in accuracy no matter how D changes. In the aspect of time cost, PTS will increasingly spend time because bitmaps' size and search space will expand as training dataset size grows. In general DBOTD performs well when D varies from 1000 K to 5000K.

DBOTD vs. TPRO and IBAT. This paragraph gives a comparison among DBOTD, TPRO and IBAT. All of the three algorithms are tested in their best

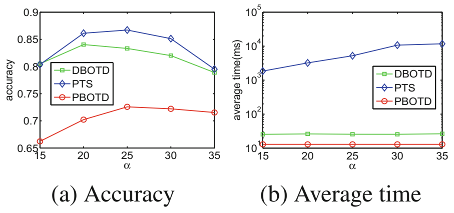


Fig. 5. Effect of α

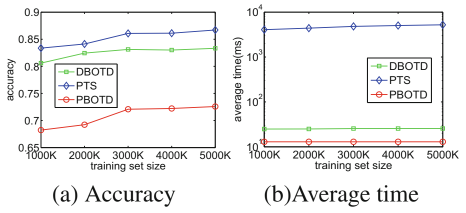


Fig. 6. Effect of D

parameters, which are listed in table 2. It is noted that we remove the attribute of time in TPRO's route distance function and such a move will not destroy the model of TPRO. And $S_{threshold}$ denotes the best threshold of outlier score in our experiment.

Table 2. Parameter setting of DBOTD, TRRO and IBAT

Algorithm	DBOTD	TPRO	IBAT
Parameters	$\alpha = 20$, $\theta = 3\%$	$k = 5$, $S_{threshold} = 115$	$m = 256$, $\psi = 100$, $S_{threshold} = 0.55$

Figure 7(a) shows the accuracy of DBOTD, TPRO and IBAT in different training data sizes. Firstly, we can see that the results of DBOTD and IBAT are almost the same in the aspect of accuracy and they both outperform TPRO greatly. It is easy to understand because the routes that TPRO chooses may be similar and then TPRO will ignore other normal routes which results in low accuracy. While DBOTD chooses routes as completely as possible and so it has a high accuracy. Secondly, it shows the difference of the average time cost in Fig. 7(b). We can know that DBOTD is slightly slower than TPRO but faster that IBAT. For all concerned, DBOTD is a great algorithm.

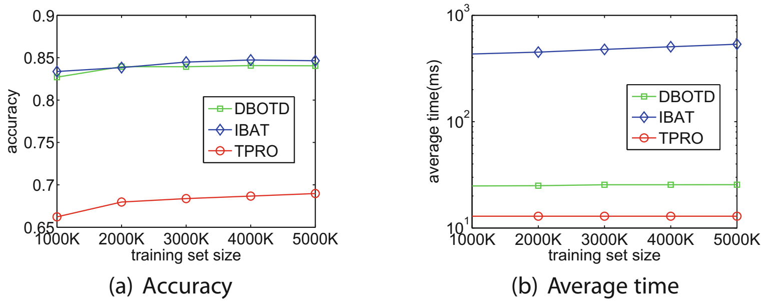


Fig. 7. Accuracy and average time cost of DBOTD, TPRO and IBAT

It can be concluded from the above experimental results that DBOTD can achieve a relatively high accuracy within short time in the algorithms that we proposed. And compared with others' algorithms, DBOTD also works best when we take both accuracy and time cost into consideration.

7 Conclusion and Future Work

In this paper, we aim at understanding historical trajectory data for discovering outlier routes of taxis. We believe that an outlier route is a route grossly different from others, meaning there are few trajectories or even no trajectory following a similar route in a dataset. Based on this idea, we firstly propose an accurate algorithm called PTS but it consumes too much time. Then we propose two trajectory clustering-based approaches PBOTD and DBOTD to cluster and extract representative routes for detecting outliers directly. Among all the three algorithms, DBOTD performs the best with a relatively high accuracy and low time cost which has been demonstrated by extensive experiment results on real datasets.

In the future, we plan to enhance our algorithm in two aspects. Firstly, we adopt transfer probability model and propose a better algorithm to optimize PTS. Secondly, we try to seek out a better cluster-based algorithm to improve accuracy.

Acknowledgement. This work was partially supported by Chinese NSFC project under grant numbers 61402312, 61232006, 61472263, 61572335, 61532018, 61502324, and Australia Research Council discovery projects under grant number DP170101172.

References

1. Chen, C., Zhang, D., Castro, P.S., Li, N., Sun, L., Li, S., Wang, Z.: iBoat: isolation-based online anomalous trajectory detection. IEEE Trans. Intell. Transp. Syst. **14**, 806–18 (2013)
2. Chen, Z., Shen, H.T., Zhou, X.: Discovering popular routes from trajectories. In: ICDE, pp. 900–911 (2011)
3. Ding, Z., Jiajie, X., Yang, Q.: Seaclouddm: a database cluster framework for managing and querying massive heterogeneous sensor sampling data. J. Supercomput. **66**, 1260–84 (2013)
4. Ester, M., Kriegel, H.-P., Sander, J., Xiaowei, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: KDD, pp. 226–231 (1994)
5. Gonzalez, H., Han, J., Li, X., Myslinska, M., Sondag, J.P.: Adaptive fastest path computation on a road network: a traffic mining approach. In: VLDB, pp. 794–805 (2007)
6. Han, J., Pei, J., Kamber, M.: Data Mining: Concepts and Techniques. Elsevier (2011)
7. Jeung, H., Yiu, M.L., Zhou, X., Jensen, C.S., Shen, H.T.: Discovery of convoys in trajectory databases. In: PVLDB, pp. 1068–1080 (2008)

8. Krumm, J., Horvitz, E.: Predestination: inferring destinations from partial trajectories. In: Dourish, P., Friday, A. (eds.) UbiComp 2006. LNCS, vol. 4206, pp. 243–60. Springer, Heidelberg (2006). doi:10.1007/11853565_15
9. Lee, J.-G., Han, J., Li, X.: Trajectory outlier detection: a partition-and-detect framework. In: ICDE, pp. 140–149 (2008)
10. Lee, J.-G., Han, J., Whang, K.-Y.: Trajectory clustering: a partition-and-group framework. In: SIGMOD, pp. 593–604 (2007)
11. Liao, L., Patterson, D.J., Fox, D., Kautz, H.: Learning, inferring transportation routines. Artif. Intell. **171**, 311–1 (2007)
12. Liu, L., Andris, C., Ratti, C.: Uncovering cabdrivers behavior patterns from their digital traces. In: Computers, Environment and Urban Systems, pp. 541–548 (2010)
13. Liu, S., Ni, L.M., Krishnan, R.: Fraud detection from taxis' driving behaviors. IEEE Trans. Veh. Technol. **63**, 464–72 (2014)
14. Lloyd, S.: Least squares quantization in PCM. IEEE Trans. Inf. Theory **28**, 129–136 (1982)
15. Masciari, E.: Trajectory outlier detection using an analytical approach. In: ICTAI, pp. 377–384 (2011)
16. Park, H.-S., Jun, C.-H.: A Simple and fast algorithm for k-meds clustering. Expert Syst. Appl. **36**, 3336–3341 (2009)
17. Pnueli, A.: Roam: Rule-and motif-based anomaly detection in massive moving object data sets. In: SDM, pp. 273–284 (2007)
18. Sacharidis, D., Patroumpas, K., Terrovitis, M., Kantere, V., Potamias, M., Mouratidis, K., Sellis, T.: On-line discovery of hot motion paths. In: EDBT, pp. 392–403 (2008)
19. Han, S., Zheng, K., Huang, J., Wang, H., Zhou, X.: Calibrating trajectory data for spatio-temporal similarity analysis. VLDB J. **24**, 93–116 (2015)
20. Lu-An Tang, Y., Zheng, J.Y., Han, J., Leung, A., Hung, C.-C., Peng, W.-C.: On discovery of traveling companions from streaming trajectories. In: ICDE, pp. 186–197 (2012)
21. Wang, H., Zheng, K., Jiajie, X., Zheng, B., Zhou, X., Sadiq, S.: Sharkdb: an in-memory column-oriented trajectory storage. In: CIKM, pp. 1409–1418 (2014)
22. Yanwei, Y., Cao, L., Rundensteiner, E.A., Wang, Q.: Detecting moving object outliers in massive-scale trajectory streams. In: KDD, pp. 422–431 (2014)
23. Zhang, D., Li, N., Zhou, Z.-H., Chen, C., Sun, L., Li, S.: iBAT: detecting anomalous taxi trajectories from gps traces. In: UbiComp, pp. 99–108 (2011)
24. Zhu, J., Jiang, W., Liu, A., Liu, G., Zhao, L.: Time-dependent popular routes based trajectory outlier detection. In: Wang, J., Cellary, W., Wang, D., Wang, H., Chen, S.-C., Li, T., Zhang, Y. (eds.) WISE 2015. LNCS, vol. 9418, pp. 16–30. Springer, Heidelberg (2015). doi:10.1007/978-3-319-26190-4_2