World Wide Web (2017) 20:111–134
DOI 10.1007/s11280-016-0400-6

Effective and efficient trajectory outlier detection based on time-dependent popular route

Jie Zhu[1] · Wei Jiang[1] · An Liu[1] · Guanfeng Liu[1] · Lei Zhao[1]

Received: 29 February 2016 / Revised: 23 June 2016 /
Accepted: 4 July 2016/ Published online: 11 August 2016

Abstract With the rapid proliferation of GPS-equipped devices, a myriad of trajectory data representing the mobility of various moving objects in two-dimensional space have been generated. This paper aims to detect the anomalous trajectories with the help of the historical trajectory dataset and the popular routes. In this paper, both of spatial and temporal abnormalities are taken into consideration simultaneously to improve the accuracy of the detection. Previous work has developed a novel time-dependent popular routes based algorithm named TPRO. TPRO focuses on finding out all outliers in the historical trajectory dataset. But in most cases, people do not care about which trajectory in the dataset is abnormal. They only yearn for the detection result of a new trajectory that is not included in the dataset. So this paper develops the the upgrade version of TPRO, named TPRRO. TPRRO is a real-time outlier detection algorithm and it contains the off-line preprocess step and the on-line detection step. In the off-line preprocess step, TTI (short for time-dependent transfer

This work was supported by the National Natural Science Foundation of China under Grant Nos. 61572335, 61572336, and 61303019, the Natural Science Foundation of Jiangsu Province of China under Grant No. BK20151223, the Natural Science Foundation of Jiangsu Provincial Department of Education of China under Grant No. 12KJB520017, and Collaborative Innovation Center of Novel Software Technology and Industrialization, Jiangsu, China.

✉ Jie Zhu
zjcomeon@gmail.com

✉ Lei Zhao
zhaol@suda.edu.cn

Wei Jiang
jwpker@outlook.com

An Liu
anliu@suda.edu.cn

Guanfeng Liu
gfliu@suda.edu.cn

[1] Soochow University, Suzhou, 215006, People's Republic of China

index) and hot TTG (short for time-dependent transfer graph) cache are constructed according to the historical trajectory dataset. Then in the on-line detection step, TTI and hot TTG cache are used to speed up the detection progress. The experiment result shows that TPRRO has a better efficiency than TPRO in detecting outliers.

Keywords Outlier detection · Time-dependent popular route · Trajectory pattern mining

1 Introduction

In recent years, the booming development of GPS-equipped portable devices has helped us in gathering a huge amount of trajectory data. According to a report of a data research organization in China, there are about 66,000 taxis in Beijing and about 1,900,000 passengers each day. Each carry generates one trajectory and there are about 69 million trajectories in one single year. Such a big dataset can help us understand the cabbies' driving behavior, the city's traffic condition and so on. On this background, extensive researchers are encouraged in trajectory pattern mining [32], such as life pattern mining [22, 25, 31], popular routes discovering [4, 21], transportation mode mining [30] and spatial item recommendation [19, 20, 23, 24].

Trajectory outlier detection (TOD) is also a popular research topic in trajectory pattern mining. According to J. Han et al. [9], an outlier means a data object that is grossly different from or inconsistent with the remaining set of data. The trajectory outlier means a trajectory that has a great difference with most other trajectories in terms of some similarity metric.

Trajectory outlier detection can be used in many practical applications. For example, it's necessary for a taxi company analyzing the historical trajectories and finding out which driver has a bad driving behavior or usually makes dishonest detours. And the taxi company can also monitor a taxi's trajectory and give an alert once an anomaly is detected.

Some TOD algorithms have been proposed. Each algorithm addresses certain aspects of abnormality. Among these TOD algorithms, the most representative methods are TRAOD (TRAjectoy Outlier Detection) [11] and IBAT (Isolation Based Anomalous Trajectory detection) [28]. TRAOD firstly splits a trajectory into many trajectory partitions and then compares each trajectory partition with its neighbors to determine whether it is an outlying portion or not. The main advantage of TRAOD lies in the ability to detect outlying sub-trajectories. But because of its sub-trajectory detection strategy, TRAOD has a high time complexity of $O(n^2)$. Moreover, the detected result of TRAOD may be influenced by irrelated trajectories because it detects outliers in the whole dataset, as shown in Figure 1. IBAT focuses on the test trajectory and tries to separate it from the reset trajectories by randomly selecting points solely from the test trajectory. IBAT is more efficient than TRAOD because IBAT does not need to partition the trajectory and the time complexity is $O(n)$. But IBAT has a same insufficiency with TRAOD: both of them do not have enough attention on the travel time (departure time, arrival time and ongoing time). TRAOD does not take the time constraint into account and IBAT just assumes the travel time of the trajectories to be detected are in the same time range but there is no in-depth analysis in IBAT.

Taking the travel time into account can ensure more accurate detection result. Figure 2 shows an example of two groups of trajectories between two areas in different time. τ_o and τ_n are two trajectories that walk the same path. But τ_o is an outlier while τ_n is not because traffic condition changes over time. In other words, outliers' pattern is not static and usually changes with the time. To detect the time-dependent outliers, previous work [34]

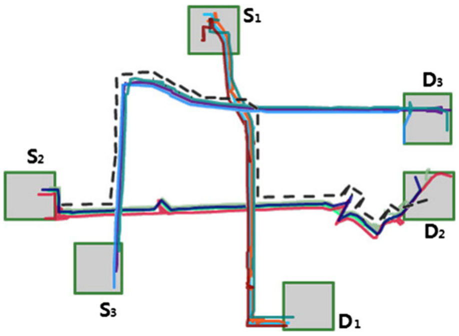


Figure 1 A set of trajectories where S_i is the source area and D_i is the destination area. The $S_2 \rightarrow D_2$ dashed curve is actually a trajectory outlier. But each subpart of it has enough closed neighbors because of being deceived by $S_1 \rightarrow D_1$ and $S_3 \rightarrow D_3$ trajectories. TRAOD can not identify this kind of outlier

has proposed a novel TOD algorithm called *time-dependent popular routes based trajectory outlier detection* (TPRO).

TPRO detects outliers with the help of the popular routes. The popular routes represent the most trajectories' pattern, so it is a reasonable solution to detect outliers based on the popular routes. As mentioned above, TPRO focuses on detecting the time-dependent outliers. So time-dependent popular routes are involved to achieve this goal. TPRO does not partition the trajectories because when facing with a large dataset, efficiency is the first priority while sub-trajectory detection is time-consuming. Given a trajectory dataset, in order to eliminate the influence of irrelevant trajectories, TPRO divides trajectories with the same source and destination (them are called relevant trajectories in this paper) into the same group. Then the dataset can be divided into many groups and detection is taken group by group. During the detection, if a trajectory has a great difference with the popular routes during its travel time, this trajectory is classified as an outlier.

TPRO focuses on finding out all outliers in the historical trajectory dataset. But in most cases, people do not care about which trajectory in the dataset is abnormal. They only submit a new trajectory that is not included in the dataset and yearn for the detection result of this trajectory. So this paper develops an upgrade version of TPRO which is called *time-dependent popular routes based* ***real-time*** *trajectory outlier detection* (TPRRO). TPRRO is a real-time detection algorithm. Given a new trajectory that is not included in the historical dataset, TPRRO can efficiently evaluate this trajectory and give a detection result according to the historical trajectory dataset.

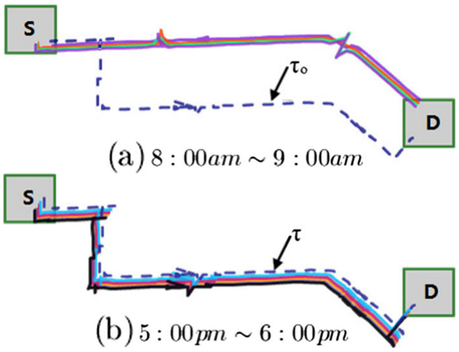


Figure 2 Two groups of trajectories which start from S and end at D in different time. τ_o is an outlier in $8:00am \sim 9:00am$ because it has a great difference with other trajectories during this time. But the traffic condition changes when $5:00pm \sim 6:00pm$. τ_n, walking the same path with τ_o, is a normal trajectory

Despite that the goal of TPRRO is easy to catch on, it is nontrivial to develop a real-time detection algorithm based on the time-dependent popular routes. There are mainly three challenges in TPRRO:

1. In TPRRO, a pending evaluated trajectory will be compared with its corresponding popular routes to judge if it is an outlier. So given a trajectory (assume its departure time is t_s and arrival time is t_d), TPRRO should efficiently retrieve the corresponding popular routes during the time of $t_s \sim t_d$.
2. TPRRO is a real-time algorithm, so it should response in seconds when evaluating a certain trajectory.
3. When calculating the difference between a trajectory and its corresponding popular routes, not only the spatial info but also the temporal info (departure time, arrival time and ongoing time) should be taken into account.

In response to these three challenge, this paper has made a lot of efforts. 1) A data structure called time-dependent transfer graph (TTG in short) is constructed in this paper. This graph records how many trajectories have passed through each road in different time. With the help of the TTG, TPRRO can efficiently retrieve the top-k most popular routes in a user specified time range. 2) To speed up detection further, this paper puts forward the time-dependent transfer index (TTI in short) and the hot TTG cache in TPRRO. With the help of the TTI and hot TTG cache, the response time is efficiently improved. 3) This paper puts forward the time-dependent edit distance to address the third challenge. The time-dependent edit distance takes not only the spatial distance but also the temporal distance into account.

The main contributions of this paper are as follows:

1. This paper presents a real-time trajectory outlier detection algorithm based on the time-dependent popular routes, which takes both spatial and temporal abnormality into consideration and gives us a novel solution in trajectory outlier detection.
2. This paper puts forward an efficient popular routes query method in this paper, which can efficiently retrieve the popular routes during a user specified time range.
3. This paper provides a real trajectory dataset in which the outliers have been labelled by user study.

The rest of this paper is organized as follows. A formal definition of our problem is given in Section 2. Sections 3 and 4 respectively give a detailed statement of TPRO and TPRRO. Section 5 shows our experiment's result. Section 6 gives a brief introduction of the related work. At last, a conclusion is given in Section 7.

2 Problem definition

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

Definition 1 (Raw Trajectory). A raw trajectory $\tilde{\tau}$ is a time-ordered sequence of sampling points: $\tilde{\tau} = (\tilde{p_1}, \tilde{p_2}, \tilde{p_3}, ..., \tilde{p_x})$. Each sampling point $\tilde{p_i}$ is represented by $\langle \tilde{l_i}, \tilde{t_i} \rangle$ where $\tilde{l_i}$ is a geographic coordinate and $\tilde{t_i}$ is the sampling time.

It is hard to find a common path from a group of raw trajectories because of the discrete sampling points. So this paper preprocesses the dataset and map each raw trajectory into the road network to get a mapped continuous trajectory.

Definition 2 (Road Network). A road network is a directed graph $G = (V, E)$ where V is a set of vertices representing road intersections and E is a set of edges representing road segments.

This paper uses v_i to represent a certain vertex in G. If v_i and v_j are two endpoints of a certain edge, then $\varphi(v_i, v_j) = 0$. If the edge's direction is $v_i \to v_j$, it can be denoted as e^i_j. Otherwise, the edge can be denoted as e^j_i.

Definition 3 (Mapped Trajectory). A mapped trajectory τ is a sequence of time-ordered road network locations. It can be denoted as $\tau = (p_1, p_2, p_3, ..., p_m)$. Each road network location p_i is represented as $\langle v_i, t_i \rangle$ where v_i is a certain vertex in the road network and for all $i \in \{1, 2, 3, ..., m-1\}$ that $\varphi(v_i, v_{i+1}) = 0$. And t_i is the time τ passing v_i.

Henceforth, this paper will only deal with the mapped trajectories. So for simplicity, this paper drops the *mapped* qualifier. Thus trajectory in the rest of the article is short for mapped trajectory. After giving a definition of the trajectory, the time-dependent route is defined as follows.

Definition 4 (Time-Dependent Route). Given a time range $[t_s, t_d]$, a $v_1 \to v_m$ time-dependent route is denoted as $\gamma = (pf_1, pf_2, pf_3, ..., pf_m)$ and each $pf_i \in \gamma$ is represented as $\langle v_i, \bar{t_i}, freq_i \rangle$ where v_i represents a certain vertex in the road network and for all $i \in \{1, 2, 3, ..., m-1\}$ that $\varphi(v_i, v_{i+1}) = 0$. Meanwhile, $freq_i$ means how many trajectories have pass through v_i in $[t_s, t_d]$ and $\bar{t_i}$ is the average pass time.

For simplicity, the *time-dependent* qualifier are dropped and route is short for time-dependent route in the rest of this paper. After giving a definition of the trajectory and the route, trajectory route distance function is put forward to indicate the difference degree between a trajectory and a route.

Definition 5 (Trajectory Route Distance Function). A trajectory route distance function $\delta(\tau, \gamma)$ is a formula that can give a difference score between τ and γ.

Based on above definitions, a formal definition of the trajectory outlier is given as following.

Definition 6 (Outlier). Given a trajectory τ, a route set $R = \{\gamma_1, \gamma_2, ..., \gamma_k\}$, a trajectory route distance function δ and an anomalous score threshold θ, the trajectory's anomalous score can be calculated in this way

$$s_\tau = \sum_{i=1}^{k} w_{\gamma_i} \cdot \delta(\tau, \gamma_i) \tag{1}$$

where w_{γ_i} is the popularity weight of γ_i among the route set R. If $s_\tau > \theta$, then τ is a θ-outlier on R and δ.

Problem Given a trajectory dataset T, a route distance function δ and an anomalous score threshold θ, this paper need to 1) find a trajectory set $T' = \{\tau_1, \tau_2, ..., \tau_n\}$ that satisfies: for all $\tau_i \in T'$, τ_i is a θ-outlier on its corresponding popular routes and δ. 2) given a new trajectory τ that is not included in T, this paper can efficiently judge if τ is an outlier based on dataset T.

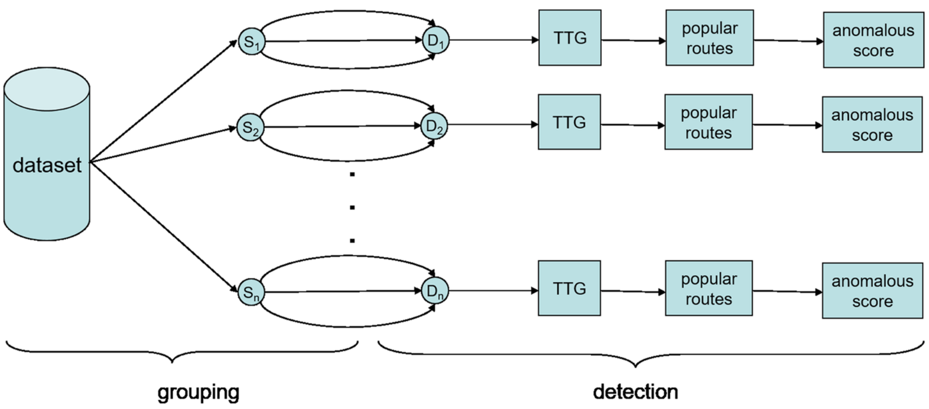


Figure 3 Overview of TPRO, which is consisted of the grouping step and the detection step

3 TPRO algorithm

TPRO can achieve the first goal in the problem proposed above. Given a trajectory dataset, to eliminate the influence of irrelevant trajectories, TPRO first divides the trajectories into the many groups according to their source and destination. Then after trajectory grouping, the detection is taken for each group respectively. In each group, TPRO firstly constructs a time-dependent transfer graph from the trajectories. Then with the help of this graph, the time-dependent popular routes querying can be more efficient. At last, a time-dependent edit distance based trajectory route distance function is proposed to judge whether a trajectory is an outlier or not. The overview of TPRO is shown in Figure 3.

3.1 Dataset grouping

The source vertex and destination vertex of a trajectory τ is represented as $\tau.s$ and $\tau.d$. If we adopt the strategy that only trajectories starting at same vertex and ending at same vertex can be gathered into one group, we find out that each group has few trajectories. So this paper develops the grid-equal-to relation to enlarge the particle size of source area and destination area.

Definition 7 (Grid-Equal-To Relation). Given two number m, n, the road network G can be split into $m \times n$ size-equal grids. For two vertices v_i, v_j, if v_i and v_j fall into the same grid, then v_i is m-n-grid-equal-to v_j. It can be denoted as $o(G, m, n, v_i, v_j) = 1$.

For two certain trajectories τ_i and τ_j, after given two grid numbers m and n, if there are $o(G, m, n, \tau_i.s, \tau_j.s) = 1$ and $o(G, m, n, \tau_i.d, \tau_j.d) = 1$, they are divided into the same group.

3.2 Construction of time-dependent transfer graph

After a certain group of trajectories with the same source and destination are mapped into the road network, a subgraph of the road network (Figure 4 shows an example of this subgraph) can be generated. And for each vertex in this subgraph, this paper develops a vertex

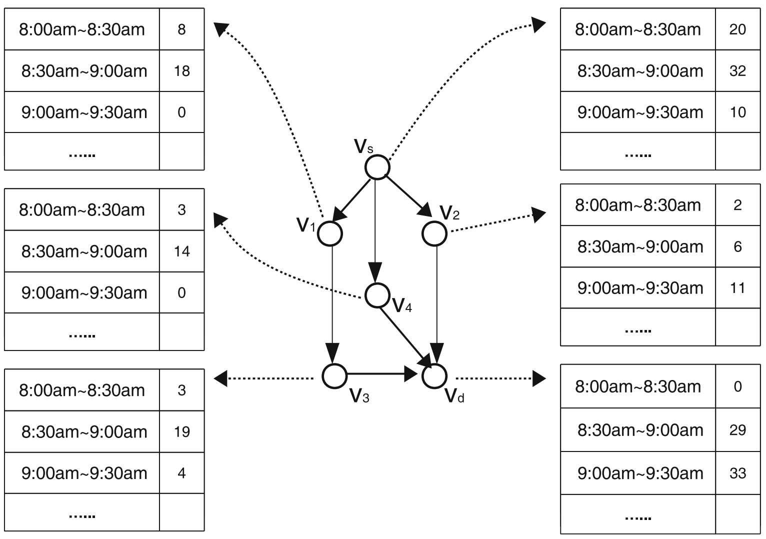


Figure 4 An example of TTG. v_s is the source and v_d is the destination. Each table beside the vertex v_i is called vertex frequency table of v_i

frequency table (i.e. the table beside each vertex in Figure 4) to record how many trajectories have pass through each vertex in different time range. This subgraph is called the time-dependent transfer graph (TTG) in this paper.

Before giving a statement of the vertex frequency table, the definition of time-equivalent-to relation will be introduced. As we known, a timestamp(such as Jan 20 2016 14 : 35 : 24) contains the date part(Jan 20 2016) and the time part(14 : 35 : 24). In this paper, TPRO drops the date part when comparing two timestamps. Because in most cases, the traffic condition take a day for a loop.

Definition 8 (Time-Equivalent-To Relation). Given two timestamps t_1, t_2 and a time distance threshold Δt, if

$$|time(t_1) - time(t_2)| \leq \Delta t \tag{2}$$

then t_1 is equivalent to t_2. Function $time(t)$ drops the date part of t and returns the time part.

In the vertex frequency tables, this paper first establishes a time distance threshold Δt. Then TPRO puts the time equivalent simpling location(vertex) together and calculates how many trajectories pass through this location(vertex) in different time range. The time distance threshold Δt is called TTG time interval in this paper.

With the help of these vertex frequency tables, TPRO can easily estimate how many trajectories have pass through a certain vertex during a user specified time range. Taking Figure 4 as an example, we can infer that there are 26 trajectories (8 trajectories during $8:00am \sim 8:30am$ and 18 trajectories during $8:30am \sim 9:00am$) have passed through v_1 during $8:00am \sim 9:00am$. In some cases, the user specified time range does not fully cover the vertex frequency table time ranges. For example, what if we want to know that how many trajectories have passed through v_1 during $8:10am \sim 9:00am$. From the TTG, we can know that there are 18 trajectories have passed through v_1 during $8:30am \sim 9:00am$, but we can not infer how many trajectories during $8:10am \sim 8:30am$ directly. In such situation, TPRO multiply the trajectories number by the proportion of the covered time range. Thus, the trajectories number during $8:10am \sim 8:30am$ is

$$8 \times \frac{8:30am - 8:10am}{8:30am - 8:00am} = 8 \times \frac{20min}{30min} \approx 5$$

Obviously, the smaller the TTG time interval is, the more accurate the inferred number is. But the space cost and time cost will increase.

And from the TTG in Figure 4, TPRO can also infer the average pass time of a certain vertex during a user specified time range. For example, there are 5 and 18 trajectories have passed through v_1 during $8:10am \sim 8:30am$ and $8:30am \sim 9:00am$ respectively. So the average pass time during $8:10am \sim 9:00am$ is

$$\frac{5 \times \frac{8:10am+8:30am}{2} + 18 \times \frac{8:30am+9:00am}{2}}{5+18} \approx 8:40am$$

3.3 Retrieving time-depended popular routes

For a trajectory τ to be tested, TPRO compares it with the popular routes during $t_s \sim t_d$ (t_s represents the departure time and t_d represents the arrival time) to judge if it is an outlier. So this paragraph explains how to query time-dependent popular routes with the help of TTG.

Assume that $t_s = 8:00am$ and $t_d = 9:00am$, TPRO should find the top-k most popular routes during this time. First of all, our algorithm can traverse the TTG and calculate each vertex's trajectories numbers and the average pass time during $t_s \sim t_d$. Then all possible routes during $t_s \sim t_d$ are figured out as follows:

- $\gamma_1 = (\langle v_s, 8:33am, 52\rangle, \langle v_1, 8:36am, 26\rangle, \langle v_3, 8:41am, 22\rangle, \langle v_d, 8:45am, 29\rangle)$
- $\gamma_2 = (\langle v_s, 8:33am, 52\rangle, \langle v_4, 8:40am, 17\rangle, \langle v_d, 8:45am, 29\rangle)$
- $\gamma_3 = (\langle v_s, 8:33am, 52\rangle, \langle v_2, 8:38am, 8\rangle, \langle v_d, 8:45am, 29\rangle)$

Now that all routes have been figured out, TPRO will judge which route is more popular. Inspired by Luo et al. [14], the route popularity and more-popular-than relation are proposed in following definitions.

Definition 9 (Route Popularity). The popularity of a certain route $\gamma = (pf_1, pf_2, pf_3, ..., pf_m)$ can be represented as an ordered frequency sequence: $\rho_\gamma = (freq_{j_1}, freq_{j_2}, freq_{j_3}, ..., freq_{j_m})$, where:

1. $\{freq_{j_1}, freq_{j_2}, freq_{j_3}, ..., freq_{j_m}\} \Leftrightarrow \{pf_1.freq,\ pf_2.freq, pf_3.freq, ..., pf_m.freq\}$
2. $freq_{j_1} \le freq_{j_2} \le ... \le freq_{j_m}$

Definition 10 (More-Popular-Than Relation). For two routes γ and $\gamma^{'}$, assume their popularity sequences are $\rho_\gamma = (freq_{j_1}, freq_{j_2}, freq_{j_3}, ..., freq_{j_m})$ and $\rho_{\gamma^{'}} = (freq^{'}_{j_1}, freq^{'}_{j_2}, freq^{'}_{j_3}, ..., freq^{'}_{j_n})$. If one of the following statements holds:

- ρ_γ is the prefix of $\rho_{\gamma^{'}}$
- or there exists a number $q \in \{1, 2, 3, ..., min(m, n)\}$ such that:
  1. $freq_{j_x} = freq^{'}_{j_x}$ for all $x \in \{1, 2, 3, ..., q-1\}$, if $q > 2$.
  2. $freq_{j_q} > freq^{'}_{j_q}$

then γ is more-popular-than $\gamma^{'}$, denoted as $\gamma \succeq \gamma'$.

According to Definition 9 and Definition 10, there are $\rho_{\gamma_1} = (22, 26, 29, 52)$, $\rho_{\gamma_2} = (17, 29, 52)$ and $\rho_{\gamma_3} = (8, 29, 52)$. Obviously, $\rho_{\gamma_1} \succeq \rho_{\gamma_2} \succeq \rho_{\gamma_3}$. It means that γ_1 is more popular than γ_2 and γ_2 is more popular than γ_3. Assume that $k = 2$, then the top-k most popular routes during $8:00am \sim 9:00am$ are γ_1 and γ_2.

Luo et al. has proved that the selected popular routes by this method satisfies three key properties: suffix-optimal (i.e., any suffix of the popular route is also popular), length-insensitive (i.e., popular does not mean the shorter/longer the better), and bottleneck-free (i.e., popular routes should not contain infrequent vertices or edges) in [14].

3.4 Outlier Detection

After the top-k popular routes have been figured out, TPRO compares the trajectory with each popular route. As we know, edit distance can represent two sequences' difference degree. But trajectory (or route) is not just a vertex sequence, it also carries the temporal information. So this paper proposes a time-dependent edit distance based trajectory route distance function to handle this problem.

Assume $\tau_{-1} = (p_1, p_2, p_3, ..., p_{m-1})$ is a sub-trajectory of $\tau = (p_1, p_2, p_3, ..., p_{m-1}, p_m)$ after removing the last point p_m. And $\gamma_{-1} = (pf_1, pf_2, pf_3, ..., pf_{n-1})$ is the prefix of $\gamma = (pf_1, pf_2, pf_3, ..., pf_{n-1}, pf_n)$ after removing the last tuple pf_n. The trajectory route distance function in TPRO is defined as a recursive equation:

$$\delta(\tau, \gamma) = Min \begin{cases} \delta(\tau_{-1}, \gamma) + delete\_cost(p_m) \\ \delta(\tau, \gamma_{-1}) + delete\_cost(pf_n) \\ \delta(\tau_{-1}, \gamma_{-1}) + replace\_cost(p_m, pf_n) \end{cases} \tag{3}$$

where

$$\begin{aligned} delete\_cost(p_m) = &\begin{cases} 0.5, & m < 2 \text{ or} p_m.v \neq p_{m-1}.v \\ 0, & otherwise \end{cases} \\ &+ \begin{cases} 0.5, & m < 2 \text{ or} |p_m.t - p_{m-1}.t| > \Delta t \\ 0, & otherwise \end{cases} \end{aligned} \tag{4}$$

$$\begin{aligned} delete\_cost(pf_n) = &\begin{cases} 0.5, & n < 2 \text{ or} pf_n.v \neq pf_{n-1}.v \\ 0, & otherwise \end{cases} \\ &+ \begin{cases} 0.5, & n < 2 \text{ or} |pf_n.\bar{t} - p_{n-1}.\bar{t}| > \Delta t \\ 0, & otherwise \end{cases} \end{aligned} \tag{5}$$

$$replace\_cost(p_m, pf_n) = \begin{cases} 1, & p_m.v \neq pf_n.v \\ 0, & otherwise \end{cases} + \begin{cases} 1, & |p_m.t - pf_n.\bar{t}| > \Delta t \\ 0, & otherwise \end{cases} \tag{6}$$

If there is only one vertex in τ (or γ), which means that $m = 1$ (or $n = 1$), then there is $\tau_{-1} = \phi$ (or $\gamma_{-1} = \phi$). Assume that the number of vertices of τ and γ are represented as $\tau.len$ and $\gamma.len$, then these two initial conditions in this recursive equation are

$$\delta(\tau, \phi) = \tau.len; \tag{7}$$

$$\delta(\phi, \gamma) = \gamma.len; \tag{8}$$

From (4), (5) and (6), we can see that the $delete\_cost$ or the $replace\_cost$ can be broken down into the spatial cost and the temporal cost. If two vertices are different, the spatial cost is 0.5 (delete) or 1 (replace). Otherwise, the spatial cost is 0. When calculating the temporal cost, TPRO has the aid of the TTG time interval Δt in Subsection 3.2. If the time lag is larger than Δt, the temporal cost is 0.5 (delete) or 1 (replace). Otherwise, the temporal cost is 0.

Algorithm 1 Trajectory Route Distance Function

```
Require: a trajectory $\tau$, a route $\gamma$
Ensure: Distance between $\tau$ and $\gamma$
 1: DECLARE int $DP[0..\tau.len][0..\gamma.len]$
 2: for $i := 0$ to $\tau.len$ do
 3:     $DP[i][0] = i$;
 4: end for
 5: for $j := 0$ to $\gamma.len$ do
 6:     $DP[0][j] = j$;
 7: end for
 8: for $i = 1$ to $\tau.len$ do
 9:     for $j = 1$ to $\gamma.len$ do
10:         $DP[i][j] = minimum($
11:             $DP[i-1][j] + delete\_cost(\tau.p_i)$,
12:             $DP[i][j-1] + delete\_cost(\gamma.pf_j)$,
13:             $DP[i-1][j-1] + replace\_cost(\tau.p_i, \gamma.pf_j)$
14:         );
15:     end for
16: end for
17: return $DP[\tau.len][\gamma.len]$;
```

The pseudo code of the time-dependent edit distance based trajectory route distance function is shown in Algorithm. 1.

The trajectory route distance function can give a difference score between a trajectory and a popular route. But in most cases, there are more than one popular route between two areas. But it does not mean that each popular route has the same popularity degree. So this paper proposes the popularity weight to represent how popular a route is among a set of routes.

Definition 11 (Popularity Weight). Assume there is a route set $R = \{\gamma_1, \gamma_2, \gamma_3, ..., \gamma_k\}$ and for each $\gamma_i = (pf_1, pf_2, pf_3, ..., pf_m) \in R$, the summary frequency of γ_i can be calculated in this way:

$$\gamma_i.sum = \sum_{a=1}^{m} \gamma_i.pf_a.freq \tag{9}$$

then the popularity weight of γ_i can be represented as

$$w_{\gamma_i} = \frac{\gamma_i.sum}{\sum_{b=1}^{k} \gamma_b.sum} \tag{10}$$

For the top-2 popular route set $R = \{\gamma_1, \gamma_2\}$ in the above example, base on Definition 11, each popular route's popularity weight is that $w_{\gamma_1} = 0.57$ and $w_{\gamma_2} = 0.43$.

3.5 Time complexity

The overall pseudo code of TPRO has been shown in Algorithm 2. From the pseudo code, we can see that the time complexity of TPRO is $O(k \cdot n)$ where k is the number of popular routes used to detect outliers and n is the number of trajectories in the dataset. In most cases, k is a small number (less than 10), so we can use approximation $O(n)$ for the time complexity.

Algorithm 2 TPRO

```
Require: road network $G$, dataset $T$, grid number $m$ and $n$, popular routes number $k$, score
    threshold $\theta$, TTG time interval $\Delta t$
Ensure: outlier set $T^{'}$
 1: //DATASET GROUPING
 2: $grid = CreateGrid(G, m, n)$; // create $m \times n$ grids on the road network
 3: $groups = \phi$;
 4: for ecah $\tau \in T$ do
 5:     $srcGrid = grid.getLocatedGrid(\tau.s)$;
 6:     $destGrid = grid.getLocatedGrid(\tau.d)$;
 7:     $groups[srcGrid, destGrid].add(\tau)$; // add trajectory to corresponding group
 8: end for
 9: //DETECTING IN EACH GROUP
10: for each $T^* \in groups$ do
11:     $ttg = CreateTTG(T^*, \Delta t)$; // construct time-dependent transfer graph
12:     for each $\tau \in T^*$ do
13:         $s_\tau = 0$;
14:         $routes = GetTopKRoutes(ttg, k, \tau.t_s, \tau.t_d)$; // query popular routes
15:         for each $\gamma \in routes$ do
16:             $s_\tau + = w_\gamma \cdot \delta(\tau, \gamma)$; // compare trajectory with each route
17:         end for
18:         if $s_\tau > \theta$ then
19:             $T^{'}.add(\tau)$; // add to outlier set
20:         end if
21:     end for
22: end for
23: return $T^{'}$;
```

4 TPRRO algorithm

Based on TPRO, this paper develops a real-time detection algorithm called TPRRO. TPRRO can successfully achieve the second goal in the problem proposed in Section 2. It means that given a new trajectory that is not included in the historical dataset, TPRRO can evaluate this trajectory and give an anomalous score about this trajectory. The overall process ends in seconds.

4.1 Overview

The overview of TPRRO is shown in Figure 5, which consists of off-line preprocessing and on-line detection step.

In the off-line preprocessing step, TPRRO builds a data structure called time-dependent transfer index(TTI) according to the historical dataset. TTI can speed up random trajectory query. In order to achieve higher efficiency, TPRRO also selects top-n hottest source and

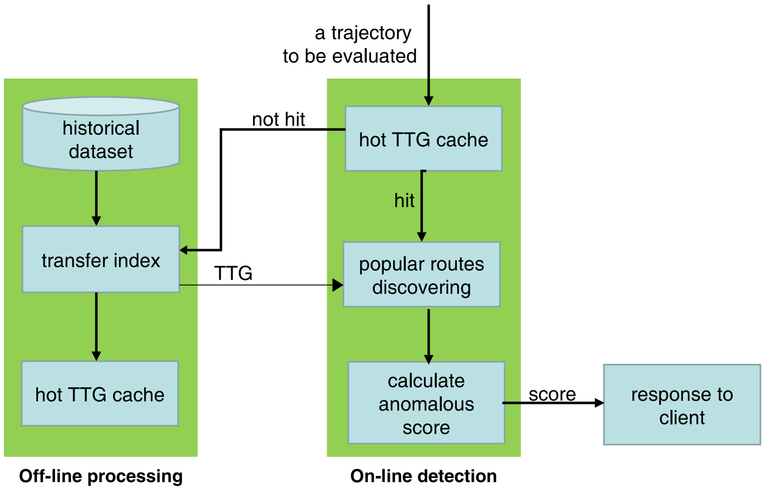


Figure 5 Overview of TPRRO, which consists of the off-line preprocessing step and on-line detection step

destination pairs and construct the TTG for each pair in advance. The pseudo code of the off-line preprocessing is shown in algorithm 3.

Algorithm 3 TPRRO Off-line Preprocessing

```
Require: road network $G$, historical dataset $T$, grid number $m$ and $n$
1: //ROAD NETWORK PARTITION
2: $grid = CreateGrid(G, m, n)$; // create $m \times n$ grids on the road network
3: //TTI CONSTRUCTION
4: $tti = \phi$; // time-dependent transfer index
5: $groupCounts = \phi$; // records how many trajectory in each source destination pair
6: for ecah $\tau \in T$ do
7:     $srcGrid = grid.getLocatedGrid(\tau.s)$;
8:     $destGrid = grid.getLocatedGrid(\tau.d)$;
9:     $groupCounts[srcGrid, destGrid].count+ = 1$;
10:    for ecah $p \in \tau$ do
11:        $grid = grid.getLocatedGrid(p)$;
12:        $tti[grid].add(\tau, p, \tau.p.t)$; // add trajectory to index
13:    end for
14: end for
15: // HOT TTG CACHE CONSTRUCTION
16: $ttgCache = \phi$;
17: $hotGroups = groupCounts.selectHotGroups()$;
18: for ecah $group \in hotGroups$ do
19:     $ttgCache.add(CreateTTG(group.src, group.dest, T))$;
20: end for
```

For a pending evaluated trajectory in the on-line detection step, TPRRO will check if the source and destination pair of this trajectory hits in the TTG cache. If hits, then TPRRO just takes the TTG from the cache. If not hits, TPRRO will construct a new TTG with the help of TTI and update the cache when the evaluation is over. After the TTG is figured out, the follow-up process is similar to the detection step in TPRO. With the help of TTG, TPRRO first retrieves the top-k most popular routes. Then the trajectory is compared with each popular route and the anomalous score is calculated. The pseudo code of the on-line detection is shown in algorithm 4.

Algorithm 4 TPRRO On-line Detection

```
Require: time-dependent transfer index $tti$, hot TTG cache $cache$, trajectory $\tau$, score threshold $\theta$
1: //TTG CONSTRUCTION
2: $ttg = cache.get(\tau.s, \tau.d)$;
3: if $ttg == null$ then
4:     $ttg = GetTTGFromTTI(\tau.s, cuurentVertex)$;
5: end if
6: //EVALUATION
7: $s_\tau = 0$;
8: $routes = GetTopKRoutes(ttg, k, \tau.t_s, \tau.t_d)$; // query popular routes
9: for each $\gamma \in routes$ do
10:     $s_\tau += w_\gamma \cdot \delta(\tau, \gamma)$; // compare trajectory with each route
11: end for
12: //RESPONSE TO CLIENT
13: $ret = 1$;
14: if $s_\tau > \theta$ then
15:     $ret = 0$;
16: end if
17: return $ret$;
```

4.2 Time-dependent Transfer Index

Time-dependent transfer index(TTI) is actually a reverse index itself. It records which trajectory has passed through which location at which time. Figure 6 shows a set of trajectories and Figure 7 shows an example of TTI that constructed from these trajectories. For a trajectory dataset, TPRRO maps each trajectory on the grid-partitioned road network. Then in each grid, TPRRO builds a B-tree like structure called tranfer B-tree. Tranfer B-tree records which trajectory has passed through this grid at which time period. By means of the TTI, TPRRO can efficiently retrieve all the trajectories that satisfy user-specified spatial and temporal constraints.

With the help of TTI, TPRRO can efficiently find out which trajectories have transferred from location loc_i to location l_j in the time range $t_i \sim t_j$. First of all, TPRRO maps loc_i and loc_j on the grid-partitioned road network, and figures out the two grids $gird_i$ and $grid_j$ that loc_i and loc_j fall into. Then TPRRO retrieves the trajectories T_i that pass through $gird_i$ in the time range $t_i \sim t_j$ by means of tranfer B-tree. In the same way, TPRRO retrieves the trajectories T_j that pass through $gird_j$ in the time range $t_i \sim t_j$. At last, TPRRO takes the intersection of T_i and T_j as the final result.

4.3 TTG cache

To speed up the progress of the on-line detection further, this paper puts forward a cache data structure called TTG cache. In the preprocessing step, this paper selects top-n hottest

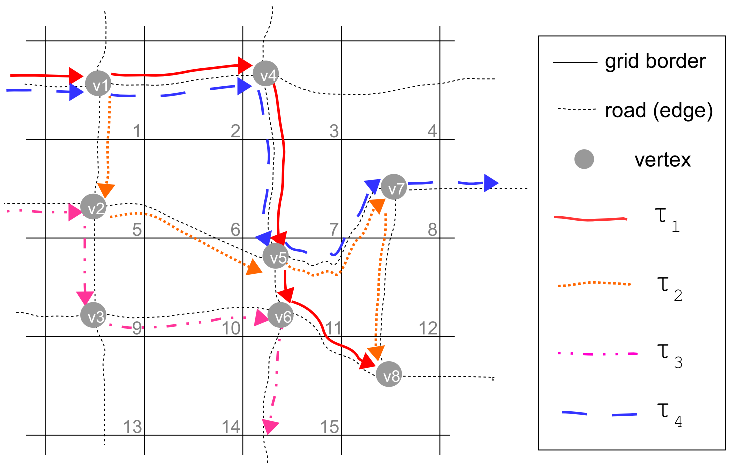


Figure 6 A set of trajectories that are mapped on the grid-partitioned road network. The legend is on the right side of the map

source and destination pairs and constructs the TTG for each pair in advance. In the progress of detection, TTG cache uses LRU-based algorithm to discard the least recently used items first.

As we know, Least Recently Used (LRU) is the most famous algorithm in cache maintainance field. This algorithm discards the least recently used items first. Figure 8 shows the procedure of LRU algorithm. The new data is added to the head of the list. And if a data is recently used, it will be moved to the head. When the list is full, the data at the tail of the list will be removed.

The overall procedure of the LRU algorithm is easy to understand and it has a good performance in most cases. But in some instances, the LRU algorithm is instable because of its update policy. For example, the capacity of LRU list is set to n, which means that there are up to n items in LRU list. Under normal circumstances, the hottest items gather around the head of the list. But if there continuously comes n infrequent items, previous hot items

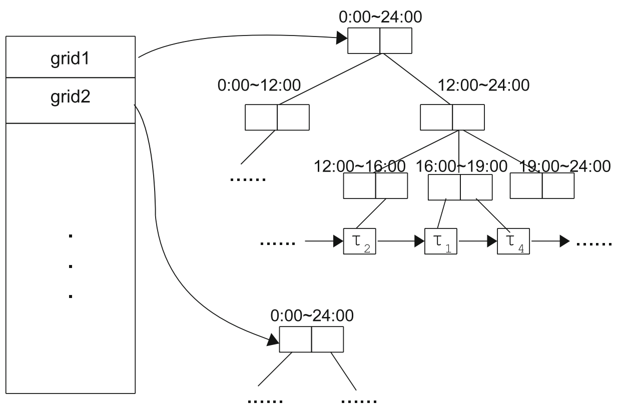


Figure 7 An example of TTI. This TTI is constructed from the trajectories in Figure 6

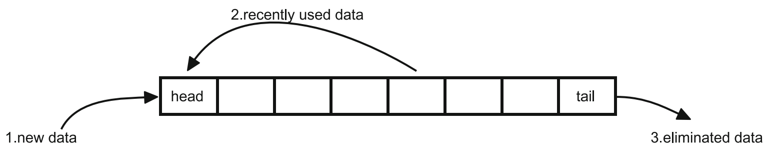


Figure 8 Procedure of the LRU algorithm

move backward and will be eliminated because the new data will be added to the head. In this situation, the head of list is occupied by the infrequent items, which takes some time for frequent items' counterattack.

Above phenomena is called Cache Pollution and to avoid this phenomena, TTG cache uses an upgraded version of the LRU algorithm, which is called Multi Queue [33]. Multi Queue(MQ) is a multi-level cache algorithm. It satisfies three key properties: 1) Minimal lifetime: hot items should stay in the cache list for at least given time $minDis$. 2) Frequency-based priority: data items should be prioritized based on their access frequencies. 3) Temporal frequency: data items that were accessed frequently in the past, but have not been accessed for a relatively long time should be replaced.

As shown in Figure 9, the MQ algorithm uses multiple LRU queues: Q_1, Q_2, ..., Q_m to maintain the cache. Items in Q_j have longer lifetime in the cache than those in $Q_i (i < j)$. MQ also uses a history buffer Q_{out} to record access frequencies of recently evicted items.

The overall procedure of the MQ algorithm is listed as follows:

1. New data item is insert at the head of Q_1.
2. In each queue Q_i, when a data item is recently used, move it to the head of this queue.
3. When the access frequency of a data item reaches a certain level, MQ will upgrade its priority. In other words, MQ deletes it from current queue Q_i and moves it to the head of next queue Q_{i+1}.

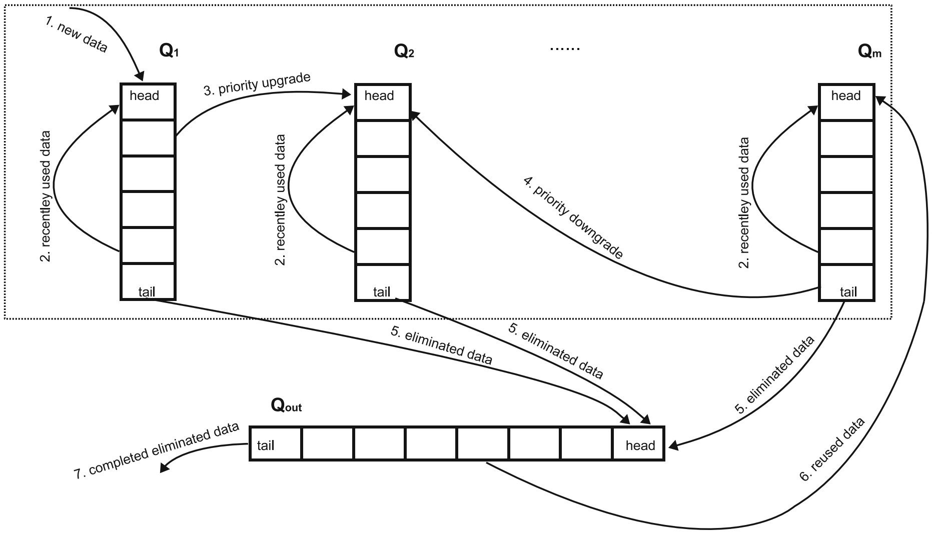


Figure 9 Procedure of the MQ algorithm

Table 1 Summary of dataset

	All dataset	Labeled dataset
Trajectories number	412,032	1,324
Car number	10,720	1,170
Spatial range	lat:[39.45 $\sim$ 40.51]	
lng:[115.71 $\sim$ 117.37]	lat:[39.79 $\sim$ 40.51]	
lng:[116.24 $\sim$ 117.36]		

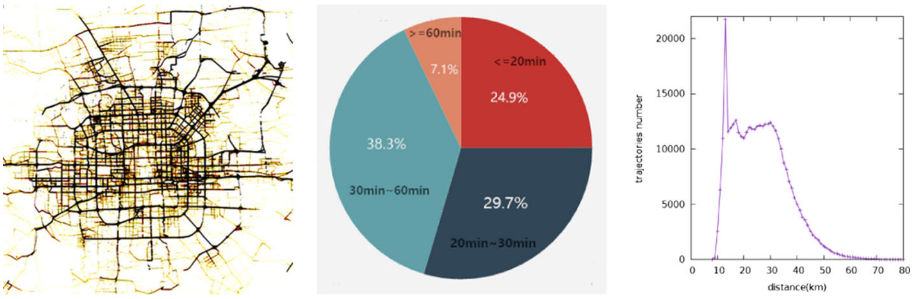
4. If a data item is not accessed in a specified time period, MQ will downgrade its priority. In other words, MQ deletes it from current queue Q_i and moves it to the head of previous queue Q_{i-1}.
5. If a queue Q_i is full, delete the data item at the tail of this queue and add this data item to the head of Q_{out}.
6. If a data item in Q_{out} is reused, delete it from Q_{out} and move it to the head of the queue where it is deleted from.
7. If Q_{out} is full, the data item at the tail completely eliminated.

5 Experiment results

This section gives an exhibition of our experiment and the results. The first subsection gives an introduction to the experiment dataset and environment setting. Then the following two subsections give an analysis on TPRO's and TPRRO's experiment results.

5.1 Experiment setting

Dataset Our experiments are taken under a real-world trajectory dataset which contains 412,032 trajectories. This dataset is collected from around 10,700 taxis in Beijing in 2012. The summary information of the dataset is shown in Table 1. Figure 10a is the visualization of this dataset. Figure 10b is the distribution of travel time and Figure 10c is the distribution of travel distance.



(a) visualization (b) distribution of travel time (c) distribution of travel distance

Figure 10 Visualization (**a**), travel time distribution (**b**) and travel distance distribution (**c**) of the real-world trajectory dataset

To evaluate the accuracy of TPRO and TPRRO, this paper also picks up about 1,300 trajectories from the dataset and asked volunteers to manually label whether each trajectory is abnormal or not. The labeling process is in the way of crowdsourcing. Each pending labelled trajectory is sent to 10 different persons to judge if it is an outlier. Only if more than 80 % of the appraisers make the same judgment, this trajectory can be concluded as abnormal or normal. Otherwise, this trajectory will be sent to another different 10 persons to rejudge. And an example is exhibited following to show how to label a trajectory in our crowdsourcing system. Assume there is a trajectory τ that needs to be labelled and its source is $\tau.s$, destination is $\tau.d$, travel time is $t_s \sim t_d$. The crowdsourcing system first retrieves all the trajectories that start from $\tau.s$ and end at $\tau.d$ during the time range $t_s \sim t_d$. Then the system maps these retrieved trajectories and the pending labelled trajectory τ on the real-world geographic map. The appraiser needs to tell if τ is an outlier. If τ travels a very different route comparing with other most trajectories, it should be labelled as an outlier.

Road Network The road network in our experiment contains about 165,000 vertices and 226,000 edges. And the road network is split into 120×130 grids in the grouping step[1]. Each grid's size is about $1.5km \times 1.5km$.

Environment Our algorithm is implemented in cpp. The machine we use to accomplish the experiment has a quadcore Inter Core i5 CPU (3.2GHz) and 8G memory. The operating system is Linux 3.13.0 x86_64 and the compiler is g++ 4.8.2.

5.2 Result of TPRO

In this part, we introduce the experiment result of TPRO in efficiency and accuracy. In next following two parts, we will first analyze how the experiment parameters affect the experiment result. Then we will give a comparison between TPRO, TRAOD and IBAT on efficiency and accuracy.

5.2.1 Varying parameters

There are mainly three parameters in the detecting step of TPRO: anomalous score threshold θ, popular routes number k and TTG time interval Δt. So this paragraph elaborates how these three parameters affect the the efficiency(process time), accuracy(detection rate and false alarm rate)[2]. Figure 11 shows how anomalous score threshold θ, popular routes number k and TTG time interval Δt affect the process time, detection rate and false alarm rate.

Efficiency The green line in Figure 12a shows how θ affects the process time when $k = 5$ and $\Delta t = 600s$. We can see that as θ increases, the process time is stable because θ only affects the final result but has no relationship with the the details of calculating process.

The green line in Figure 12b shows how k affects the process time when $\theta = 1.0$ and $\Delta t = 600s$. It shows that as k increases, the process time will go up linearly. Because k

[1]That's to say m is set to 120 and n is set to 130 in the grouping step.

[2]These three evaluating indicators are counted under the labeled dataset.

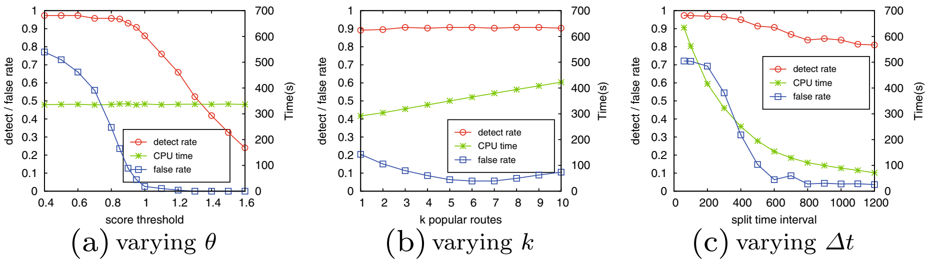


Figure 11 Detection rate, false alarm rate and time cost under varying θ, k and Δt

represents how many popular routes will be used to judge if a trajectory is an outlier and each trajectory will be compared with each route. Of course, the more popular routes are used, the more time-consuming it will be.

The green line in Figure 12c shows how Δt affects the process time when $k = 5$ and $\theta = 1.0$. The more smaller the Δt is, the more specific the TTG vertex frequency table is. But the popular routes query time will be longer. So as Δt increases, the process time falls.

Accuracy Figure 12a shows how θ affects the detection rate(red line), false alarm rate(blue line) when $k = 5$ and $\Delta t = 600s$. As θ increases, which means that the detection criterion becoming more conservative, the detection rate and the false alarm rate will fall. When $\theta \approx 1.0$, we have a high detection rate and a low false alarm rate.

Figure 12b shows how k affects the detection rate(red line), false alarm rate(blue line) when $\theta = 1.0$ and $\Delta t = 600s$. It shows that as k increases, the false alarm rate will fall. Because the more popular routes are used, the more accurate the result will be. But in most cases, there is only one popular route between two areas, so k has a small effect on the detection rate. On the contrary, if we use too many top-k popular routes, the false alarm will go up because there are not so many popular routes between two areas. When $k = 5$, we can have a high detection rate and a low false alarm rate.

Figure 12c shows how Δt affects the detection rate(red line), false alarm rate(blue line) when $k = 5$ and $\theta = 1.0$. If Δt is too small, TPRO will overstate the temporal cost when calculating the distance between a trajectory and a route. So the false alarm rate is very high and will fall as Δt increases. But smaller Δt leads to more accurate selected popular routes, which results in higher detection rate. When $\Delta t = 600s$, we can have a low false alarm rate and a less process time.

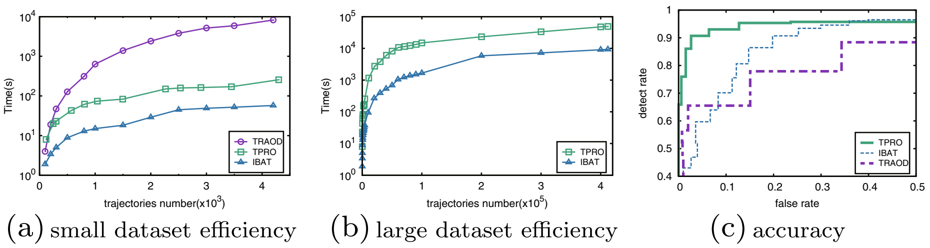


Figure 12 Efficiency under the small dataset (**a**) and the large dataset (**b**). ROC curves of TPRO, TRAOD and IBAT (**c**)

Table 2 Parameter setting of TPRO, TRAOD and IBAT

Algorithm	Parameters
TPRO	$k = 5$, $\Delta t = 600s$
TRAOD	$D = 80$, $p = 0.95$
IBAT	$m = 100$, $\psi = 256$

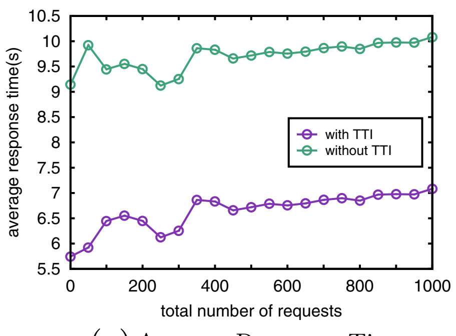
5.2.2 TPRO vs. TRAOD and IBAT

This paragraph gives a comparison between TPRO, TRAOD(group-and-partition method) and IBAT(isolation based method). All of the three algorithms are tested in their best parameters, which are listed in Table 2. The contrastive experiment results are shown in Figure 12, which consist of efficiency comparison and accuracy comparison.

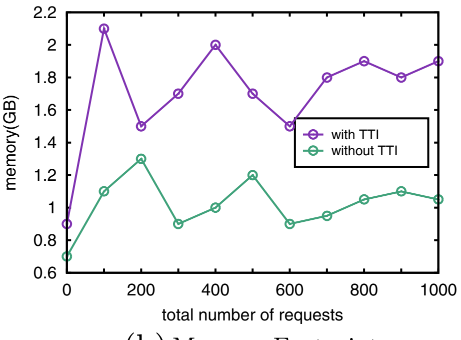
Efficiency Figure 13a and b show the process time of TPRO, TRAOD and IBAT under different scale datasets. Because the time complexity of TRAOD is $O(n^2)$, which is very time consuming in larger dataset detection, we only test it on the small dataset. From these two figures, we can see that the time cost of TPRO is between IBAT's and TRAOD's.

Accuracy In practice, detection rate (the fraction of anomalous trajectories that are successfully detected) and false alarm rate (the fraction of normal ones that are predicted to be anomalous) are two important measures to evaluate the performance of an anomaly detection method. Obviously, a good outlier detection method should have a high detection rate and a low false alarm rate. After we plot the detection rate on y-axis and the false alarm rate on x-axis, we can get a curve called Receiver Operating Characteristic (ROC) [5] curve. The AUC [1] value is defined as the area under the ROC curve. For a randomly chosen normal trajectory τ_n and a randomly chosen anomalous trajectory τ_a, the AUC value is equal to the probability that $s_{\tau_a} > s_{\tau_n}$. Obviously, if the AUC value is close to 1, the outlier detection method is of high quality.

Figure 13c shows the ROC curves of TPRO, TRAOD and IBAT. For better illustration, the ranges of false alarm rate and detection rate are set to $[0 \sim 0.5]$ and $[0.4 \sim 1]$. We can see that TPRO has a larger area under the ROC curve than TRAOD and IBAT. It means that TPRO has a better performance than TRAOD and IBAT in accuracy.

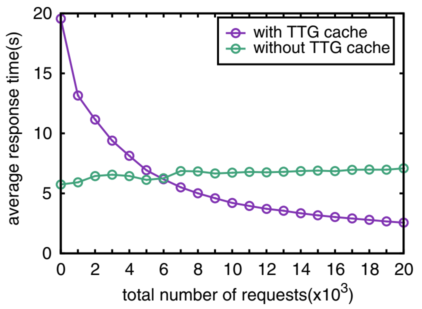


(a) Average Response Time

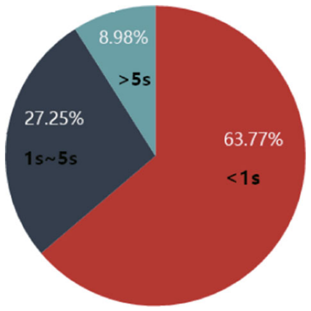


(b) Memory Footprint

Figure 13 Average response time (**a**) and memory footprint (**b**) with TTI(*purple line*) and without TTI(*green line*)



(a) Average Response Time



(b) Response Time Distribution

Figure 14 **a** average response time with TTG cache (*purple line*) and without TTG cache (*green line*). **b** distribution of response time

5.3 Result of TPRRO

As we mentioned in Section 4, thanks to TTI and TTG cache, TPRRO makes some improvement in efficiency compared to TPRO. So this part gives a statement of the experiment result of TPRRO. The first subpart analyzes the effectiveness of TTI and TTG cache. Then the second subpart gives a comparison between TPRRO and IBOAT (the real-time version of IBAT).

5.3.1 Effectiveness analysis

Efficiency This paragraph gives a specific statement of how much TTI and TTG cache can speed up the detection progress. This paper takes four comparative experiments to demonstrate the effectiveness of TTI and TTG cache.

Figure 13a shows the effectiveness in average response time caused by TTI. The purple line shows the average response time that uses TTI and the green line shows the result that doesn't use TTI. And Figure 13b shows the memory footprint of TPRRO with TTI (purple line) and without TTI(green line). These two figures show that TTI can greatly reduce the response time and it only consumes a little more memory.

Figure 14a shows the average response time with TTG cache (purple line) and without TTG cache(green line). It shows that the average response time with TTG cache is very high at first, because TTG cache is in its initial state and the cache hit ratio is very low. But as the algorithm running, TTG cache tends to be perfect gradually, so the response time falls. When the algorithm has processed about 6,000 detection requests, the average response time with TTG cache is equal to the average response time without TTG cache. As the algorithm continues running, the average response time with TTG cache further falls and approaches to the level of $2 \sim 3$s. Figure 14b shows the distribution of each detection request's response time. It shows that 60 % requests are processed below one second, which means that the TTG cache can greatly reduce the response time. Figure 15 shows the tendency of TTG cache hit ratio as TPRRO is running.

Accuracy As we mentioned above, TPRRO is a real-time version of TPRO. The main detection idea of TPRRO is same to TPRO's, namely, detecting outliers based on the time-dependent popular routes. TPRRO makes some improvements in efficiency but the

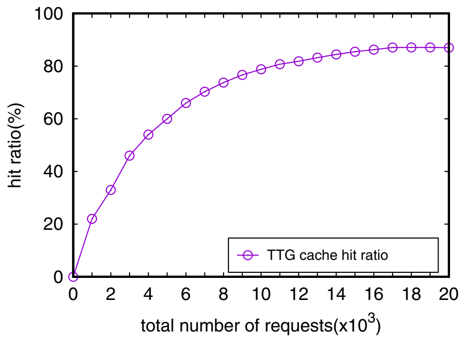


Figure 15 Tendency of cache hit ratio as the number of detection request rising. It shows that the cache hit ratio increases as receiving more detection requests. But the cache hit ratio will be stable after receiving enough detection requests

evaluation criteria of outlier is unaltered. Sections 5.2.1 and 5.2.2 has elaborated the accuracy of TPRO, so this section does not belabour the accuracy of TPRRO.

5.3.2 *TPRRO vs. IBOAT*

To further elaborate the effectiveness of TPRRO, this paper takes a comparative experiment with IBOAT. IBOAT is a real-time version of IBAT and it is of high efficiency. Figure 16 shows the tendency of the average response time as algorithms running. It shows that the average response time of TPRRO is very high at first, but as TPRRO running, TTG cache tends to be perfect gradually and the response time falls. When TPRRO has processed about 16,000 requests, the TTG cache hit ratio becomes stable and TPRRO's average response time is equal to IBOAT's. It shows that TPRRO is also of high efficiency. TPRRO and IBOAT are two real-time algorithms that originate in TPRO and IBAT, respectively. The main idea of TPRRO and TPRO (or IBOAT and IBAT) is same. Section 5.2.2 has elaborated the accuracy comparison of TPRO and IBAT, so this section does not belabour the accuracy comparison of TPRRO and IBOAT.

6 Related work

Some related works are introduced in this part, which can be categorized into two groups. The first one focuses on trajectory outlier detection and the second one focuses on popular route mining.

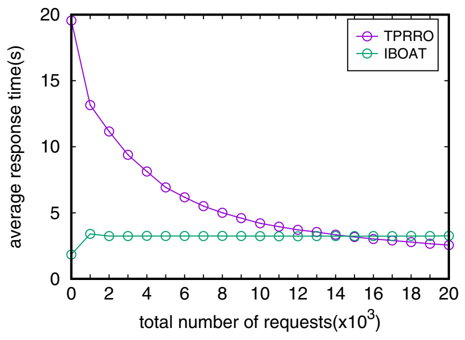


Figure 16 The average response time of TPRRO is very high at first, but as TPRRO running, the response time falls. When TPRRO has processed about 16,000 requests, TPRRO's average response time is equal to IBOAT's

Trajectory Outlier Detection Some algorithms have been proposed to detect trajectory outlier, but each addresses certain aspects of abnormality. Lee et al. [11] put forward a group-and-detect framework and develop an algorithm called TRAOD. TRAOD splits a trajectory into various subparts (at equal intervals), then a hybrid of the distance-based and density-based approach is used to classify each subpart is abnormal or not. Zhang et al. [28] propose an isolation based method, called IBAT. They focus on the test trajectory, and try to separate it from the reset trajectories by randomly selecting points solely from the test trajectory. For a group of trajectories and a trajectory will be tested in this group, they randomly pick a point from the test trajectory and remove other trajectories which do not contain this point. This process is repeated until no trajectory is left or all the trajectories left contain all the points the test trajectory has. If the test trajectory is an outlier, this process will end very soon. And Li et al. [13] emphasize on historical similarity trends between data points. At each time step, each road segment checks its similarity with the other road segments, and the historical similarity values are recorded in a temporal neighborhood vector at each road segment. Outliers are calculated from drastic changes in these vectors [8]. Guan et al. [27] use a feature vector, such as $\langle direction, speed, angle, location \rangle$, to represent a trajectory segment and detect the outliers according to these features. Mohamad et al. [15] take the speed and turn directions into consideration. If a trajectory has a sudden speed change or some unexpected turns, it is an abnormal trajectory. And there are also some studies have used learning methods to identify anomalous trajectories [12, 18]. But these methods usually need training data, which is inconvenient to label. Recently, Zhang et al. [29] combine multi-factors into outlier detection to find more meaningful trajectory outliers. They resort to Canonical Correlation Analysis (CCA) to optimize the number of factors when determining what factors will be considered. In [2], they propose data structures and algorithms employing local clustering and piecewise VP-tree based rescheduling to efficiently conduct such a task. Ge et al. [6] compute a score based on the evolving moving direction and density of trajectories, and make use of a decay function to include previous scores. In [13], they identify outlier road segments by detecting drastic changes between current data and historical trends. And Chen et al. [3] believe that anomalous trajectories are few and different. They use the idea of isolating trajectories and adopt an adaptive working window of the latest incoming GPS points to compare against the set of historical trajectories. In [26], they propose novel neighbor-based trajectory outlier definitions. Furthermore they design an optimized MEX strategy scalable to big data trajectory streams to detect the new classes of outliers, rendering moving object outliers detection practical in real time applications.

Popular Route Mining Finding the most desirable path has been a hot research topic for decades. Many works [7, 10, 16] have been done in finding the shortest/fastest path. But the popular route does not mean the shortest or fastest path. In most cases, we prefer the most frequent path as the popular route. Lots of algorithms have been proposed for popular route searching. Zaiben et al. [4] introduce a transfer probability network to discover popular route from historical trajectories. They derive the probability of transferring from every significant location to the destination based on the historical trajectories, and the transfer probability is used as an indicator of popularity. The popularity of a route is defined as the product of transfer probabilities of all significant locations on the route. Luo et al. [14] also construct a network graph (called footmark graph) to mine frequent path. But they describe the edge frequency as the total number of trajectories passing through the edge. Then they define a descending edge frequency sequence to judge which path is more frequent. Another work, such as [21], aims at deriving routes from uncertain trajectory data. And in [17], they use mobile gps data to generate the on-line heat maps of popular routes.

7 Conclusions

In this paper, we propose a time-dependent popular routes based real-time outlier detection algorithms named TPRRO. It takes spatial and temporal abnormality into consideration. TPRRO is an upgrade version of TPRO. TPRO focuses on finding out all outliers in the historical trajectory dataset. But in most cases, people do not care about which trajectory in the dataset is abnormal. They only yearn for the detection result of a new trajectory that is not included in the dataset. TPRRO can address this problem and it contains the off-line preprocess step and the on-line detection step. In the off-line preprocess step, TTI and hot TTG cache are constructed according to the historical trajectory dataset. Then in the on-line detection step, TTI and hot TTG cache are used to speed up the detection progress. The experiment result shows that TPRRO has a better efficiency than TPRO in detecting outliers.

In the future, we plan to enhance our algorithm in two directions. Firstly, we are going to develop an algorithm that focuses on sub-trajectory detection based on time-dependent popular routes. Secondly, we want to develop a outlier detection system based on TPRO and TPRRO.

References

1. Bradley, A.P.: The use of the area under the roc curve in the evaluation of machine learning algorithms. Pattern Recogn. **30**(7), 1145–1159 (1997)
2. Bu, Y., Chen, L., Fu, A.W.C., Liu, D.: Efficient anomaly monitoring over moving object trajectory streams. In: ACM KDD, pp. 159–168 (2009)
3. Chen, C., Zhang, D., Castro, P.S., Li, N., Sun, L., Li, S., Wang, Z.: iboat: Isolation-based online anomalous trajectory detection (2013)
4. Chen, Z., Shen, H.T., Zhou, X.: Discovering popular routes from trajectories. In: IEEE ICDE, pp. 900–911 (2011)
5. Fawcett, T.: An introduction to roc analysis. Pattern Recogn. Lett. **27**(8), 861–874 (2006)
6. Ge, Y., Xiong, H., Zhou, Z.H., Ozdemir, H., Yu, J., Lee, K.C.: Top-eye: Top-k evolving trajectory outlier detection. In: Proceedings of the 19th ACM international conference on Information and knowledge management, pp. 1733-?1736. ACM (2010)
7. 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 (2007)
8. Gupta, M., Gao, J., Aggarwal, C., Han, J.: Outlier detection for temporal data. Synthesis Lectures on Data Mining and Knowledge Discovery pp. 1–129 (2014)
9. Han, J., Kamber, M., Pei, J.: Data mining: concepts and techniques. Morgan Kaufmann Publishers (2006)
10. Kanoulas, E., Du, Y., Xia, T., Zhang, D.: Finding fastest paths on a road network with speed patterns. In: IEEE ICDE, pp. 10–10 (2006)
11. Lee, J.G., Han, J., Li, X.: Trajectory outlier detection: a partition-and-detect framework. In: IEEE ICDE, pp. 140–149 (2008)
12. Li, X., Han, J., Kim, S., Gonzalez, H.: Roam: Rule-and motif-based anomaly detection in massive moving object data sets. In: SIAM SDM, pp. 273–284 (2007)
13. Li, X., Li, Z., Han, J., Lee, J.G.: Temporal outlier detection in vehicle traffic data. In: IEEE ICDE, pp. 1319–1322 (2009)
14. Luo, W., Tan, H., Chen, L., Ni, L.M.: Finding time period-based most frequent path in big trajectory data. In: ACM SIGMOD, pp. 713–724 (2013)
15. Mohamad, I., Ali, M., Ismail, M.: Abnormal driving detection using real time global positioning system data. In: Space Science and Communication (Iconspace), pp. 1–6. IEEE (2011)
16. Sacharidis, D., Patroumpas, K., Terrovitis, M., Kantere, V., Potamias, M., Mouratidis, K., Sellis, T.: On-line discovery of hot motion paths. In: ACM EDBT (2008)
17. Sainio, J., Westerholm, J., Oksanen, J.: Generating heat maps of popular routes online from massive mobile sports tracking application data in milliseconds while respecting privacy. ISPRS Int. J. Geo-Inf. **4**(4), 1813–1826 (2015)

18. Sillito, R.R., Fisher, R.B.: Semi-supervised learning for anomalous trajectory detection. In: BMVC, pp. 1–10 (2008)
19. Wang, W., Yin, H., Chen, L., Sun, Y., Sadiq, S., Zhou, X.: Geo-sage: a geographical sparse additive generative model for spatial item recommendation. In: Proceedings of the 21Th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 1255–1264. ACM (2015)
20. Wang, W., Yin, H., Sadiq, S., Chen, L., Xie, M., Zhou, X.: Spore: A sequential personalized spatial item recommender system
21. Wei, L.Y., Zheng, Y., Peng, W.C.: Constructing popular routes from uncertain trajectories. In: ACM SIGKDD, pp. 195–203 (2012)
22. Ye, Y., Zheng, Y., Chen, Y., Feng, J., Xie, X.: Mining individual life pattern based on location history. In: IEEE MDM, pp. 1–10 (2009)
23. Yin, H., Cui, B., Chen, L., Hu, Z., Zhang, C.: Modeling location-based user rating profiles for personalized recommendation. ACM Trans. Knowl. Discov. Data (TKDD) **9**(3), 19 (2015)
24. Yin, H., Cui, B., Sun, Y., Hu, Z., Chen, L.: Lcars: a spatial item recommender system. ACM Trans. Inf. Syst. (TOIS) **32**(3), 11 (2014)
25. Yin, H., Hu, Z., Zhou, X., Wang, H., Zheng, K., Nguyen, Q.V.H., Sadiq, S.: Discovering interpretable geo-social communities for user behavior prediction
26. Yu, Y., Cao, L., Rundensteiner, E.A., Wang, Q.: Detecting moving object outliers in massive-scale trajectory streams. In: ACM KDD, pp. 422–431 (2014)
27. Yuan, G., Xia, S., Zhang, L., Zhou, Y., Ji, C.: Trajectory outlier detection algorithm based on structural features. J. Comput. Inf. Syst. **7**(11), 4137–4144 (2011)
28. Zhang, D., Li, N., Zhou, Z.H., Chen, C., Sun, L., Li, S.: Ibat: Detecting Anomalous Taxi Trajectories from Gps Traces. In: ACM Ubicomp, pp. 99–108 (2011)
29. Zhang, L., Zimu, H., Guang, Y.: Trajectory outlier detection based on multi-factors. IEICE Trans. Inf. Syst. **97**(8), 2170–2173 (2014)
30. Zheng, Y., Liu, L., Wang, L., Xie, X.: Learning transportation mode from raw Gps data for geographic applications on the WEB. In: WWW, pp. 247–256 (2008)
31. Zheng, Y., Xie, X., Ma, W.Y.: Geolife: a collaborative social networking service among user, location and trajectory. IEEE Data Engineering Bulletin, 32–39 (2010)
32. Zheng, Y., Zhou, X.: Computing with spatial trajectories. Springer Science & Business Media (2011)
33. Zhou, Y., Philbin, J., Li, K.: The multi-queue replacement algorithm for second level buffer caches. In: USENIX Annual Technical Conference, General Track, pp. 91–104 (2001)
34. Zhu, J., Jiang, W., Liu, A., Liu, G., Zhao, L.: Time-dependent popular routes based trajectory outlier detection. In: WEB Information Systems Engineering–WISE 2015, pp. 16–30. Springer (2015)