Hindawi
Wireless Communications and Mobile Computing
Volume 2019, Article ID 7803293, 13 pages
https://doi.org/10.1155/2019/7803293

Research Article

A Trajectory Partition Method Based on Combined Movement Features

Ji Tang,[1] Linfeng Liu ,[1,2] and Jiagao Wu[1]

[1]*Jiangsu Key Laboratory of Big Data Security and Intelligent Processing, Nanjing University of Posts and Telecommunications, Nanjing 210023, China*
[2]*School of Computer and Information, Hohai University, Nanjing 211100, China*

Correspondence should be addressed to Linfeng Liu; liulinfeng@gmail.com

Received 24 February 2019; Revised 19 June 2019; Accepted 15 July 2019; Published 24 July 2019

Academic Editor: Rüdiger C. Pryss

Trajectory data mining has become an increasing concern in the location-based applications, and the trajectory partition is taken as the primary procedure of trajectory data mining. The amount of movement trajectories of nodes is typically very large, and the trajectory shapes are extremely diverse, which makes the trajectory partition a vital issue to the trajectory data mining results. In this work, the movement behaviors of nodes are analyzed from the aspects of moving speeds, stop points, and moving directions, and then a novel Trajectory Partition Method based on combined movement Features (TPMF) is proposed to partition the trajectories. In TPMF, we first extract the change points where the movement speeds of nodes are varied significantly; then, we extract the stop points by detecting the speed variations of nodes; finally, the Douglas-Peucker algorithm is applied to partition the subtrajectories according to the extracted feature points (change points and stop points). Simulations are carried out on the *Geolife* trajectory dataset, and the simulation results indicate that TPMF can achieve a preferable trade-off between the simplification rate and the trajectory partition error, while the running time is shortened as well.

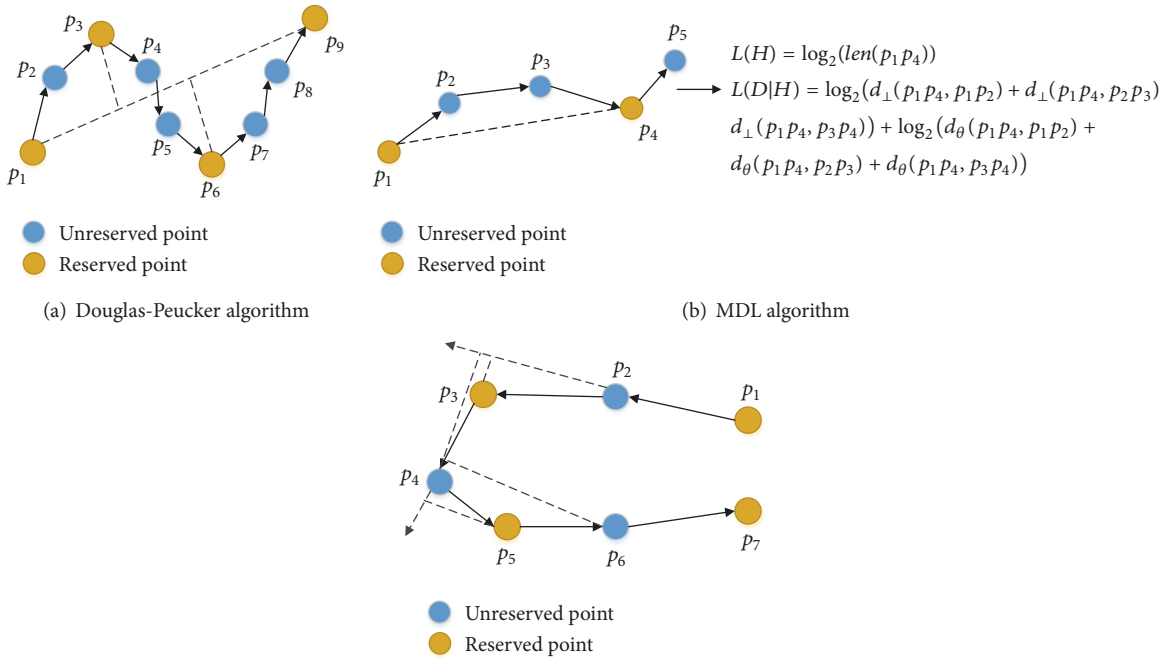
1. Introduction

Recently, the mobile communication devices with GPS modules are very popular due to the development of location-aware technology. For example, some people (nodes) carrying communication devices travel along urban roads, and the personal geographical locations can be recorded by the GPS modules at regular intervals. Thus, each movement trajectory consisted of some GPS points arranged in chronological order. Note that we can obtain many valuable information by mining these trajectories, such as the movement behaviors of nodes, which can be exploited and served for the applications of location recommendations [1], destination predictions [2], and personal navigations [3].

In general, the personal trajectories are very complicated, since the nodes always move casually. There is a vital issue to be addressed for the trajectory data mining: the trajectory data is generated quickly, and the storage space of nodes is occupied, which brings large computation burdens and storage burdens in trajectory data mining. On the contrary, the reduction of the number of trajectory data leads to the loss of some key information, making the outcome of trajectory prediction inaccurate.

The trajectory partition is served for the removals of redundant GPS points and the detections of node behaviors. Especially, it is necessary to simplify the trajectories to avoid the unbearable computation burdens in trajectory data mining phase, while the valuable information regarding the trajectory contours can be reserved as much as possible. In this paper, we partition the trajectories into successive line segments according to the combined movement features (moving speeds, stop points, and moving directions). The main intention of our work is to make a preferable trade-off between the simplification rate and the trajectory partition error while shortening the running time.

To reduce the computation burdens in the trajectory mining phase, the existing trajectory partition methods always attempt to find and remove the redundant trajectory points and reserve the valuable trajectory points. However, the results of trajectory partition are not preferable because



(a) Douglas-Peucker algorithm

(b) MDL algorithm

(c) Algorithm based on position perturbation

Figure 1: Trajectory compression and partition.

the movement features are not comprehensively considered in these methods. To this end, we take into account the typical movement features, such as the spatial locations, directions, time, movements, and stops for analyzing the trajectories, and these features are concluded and combined as moving speeds, stop points, and moving directions. The innovations of this paper are given as follows:

(1) The novel idea of partitioning the trajectories based on moving speeds, stop points, and moving directions is proposed to remove the redundant trajectory points while ensuring the trajectory accuracy.

(2) A stop point extraction algorithm based on moving speeds is put forward to extract the stop points more accurately. Especially, we improve the stop point extraction through applying our proposed mechanisms of forward search and backward search based on moving speeds. Our method reduces the unnecessary computations by detecting the speeds, and more accurate results can be obtained from the mechanisms of forward search and backward search.

(3) A Douglas-Peucker algorithm based on perpendicular Euclidean distance and directions is utilized to maintain the overall shapes of trajectories as much as possible through retaining the points where the directions change abruptly; i.e., we combine the indices of perpendicular Euclidean distance and moving directions to identify the points where the directions change abruptly while the overall contours of trajectories are basically maintained.

The rest of the paper is organized as follows: Section 2 gives some related works. Section 3 introduces the Trajectory Partition Method based on combined movement Features (TPMF). Section 4 reports the simulation results for performance evaluation of TPMF. Finally, Section 5 concludes this paper.

2. Related Work

The technique of trajectory partition is derived from that of trajectory compression. In early work, two representative strategies of trajectory compression are proposed in [4], which introduces an offline compression method and an online compression method, respectively. Given a fully generated trajectory, the offline compression generates an approximate trajectory by removing some redundant points from the original trajectory. The well-known offline trajectory compression method is Douglas-Peucker (DP) algorithm [5], and the idea of DP is to replace a subtrajectory with an approximate line segment, as shown in Figure 1(a). DP recursively partitions each trajectory into two subtrajectories by selecting the points can contributing the largest error as the splitting points, until the specified perpendicular Euclidean distance is satisfied. As the perpendicular Euclidean distance is not associated with the time stamps, and thus an improved Top-Down algorithm [6] is proposed to compress the trajectories, where the temporal-distance information of each sample point is defined and considered. In contrast to the offline compression, the online compression does not require complete trajectories, and each trajectory is compressed immediately while it is transmitted. SQUISH [7] works in

the streaming environment with a fixed-size buffer. For each point, it maintains a priority which serves as the upper bound of SED distance for the neighboring points. This is because when a point is deleted from buffer, its priority score will be accumulated to its two neighboring points. Since SQUISH is not error-bounded, its subsequent work SQUISH-E [8] is designed to be adaptive to different objectives by introducing two parameters λ (compression ratio bound) and μ (compression error bound). BQS [9] picks at most eight significant points, forming a convex hull to enclose all the points in the buffer. Then, an upper bound and a lower bound are derived, such that a point can be quickly decided to be removed or reserved. Recently, Liu et al. propose an one-pass error bounded trajectory simplification algorithm named OPERB [10]. Based on a local distance checking method, OPERB maintains a directed line segment to approximate the buffered points and guarantees that the distance from the current point to the line segment is bounded.

In addition, some other researches aim to reduce the number of trajectory data and maintain the semantic meanings, such as the types of places where the nodes stay for a long period. References [11, 12] attempt to maintain the semantic meanings of original trajectories after trajectory compression process.

As a basis of the techniques of trajectory clustering and trajectory classification, we need to partition the original trajectories into several successive line segments for a further process. Lee et al. [13] present a trajectory partition method based on Minimum Description Length (MDL), as shown in Figure 1(b), where $L(H)$ represents the length sum of all trajectory partitions, and $L(D|H)$ represents the sum of the difference between the trajectory and its trajectory partitions. MDL finds a list of characteristic points to minimize the value of $L(H) + L(D|H)$. Reference [14] provides a trajectory partition method based on the position perturbation, as shown in Figure 1(c), and this method calculates the perpendicular distance from a point to the previous moving direction. If the perpendicular distance is larger than a preset threshold, then the previous point is reserved. The above steps are repeated until all points have been traversed. Besides, Fu et al. [15] first find the stop points and then apply DP algorithm to simplify the trajectory segments between adjacent stop points. Besides, a corner detection based method is adopted to divide trajectories in [16].

Most of aforementioned works always divide trajectories spatially, and it is necessary to take into account the critical movement features such as the moving speeds, stop points, and moving directions. Moreover, a preferable trade-off between the simplification rate and the trajectory partition error is expected to be achieved.

3. Trajectory Partition Method

In real-world scenes, we note the three significant characteristics regarding the node movements: (*i*) the moving speeds of nodes changes greatly due to the switches of different travel modes or some special events (e.g., sudden braking or abrupt deceleration in driving) [17]; (*ii*) the nodes are prone to visit some locations intentionally or unintentionally, and the locations visited frequently or stayed for a long period can be marked as stop points; (*iii*) the moving directions of nodes are varied frequently when some travel modes (such as the walking mode) are adopted [17].

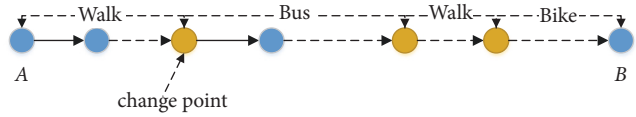


Figure 2: Travel mode transition.

To this end, we propose a trajectory partition method based on combined movement features (moving speeds, stop points and moving directions). Several definitions are first given as follows.

Definition 1 (a GPS point). A GPS point $p_i = (lat_i, lng_i, t_i)$ is expressed as a quadruple, which is composed of the latitude, longitude and timestamp of the i-th point, where $i = 1, 2, \ldots, n$, n denoting the number of GPS points in a trajectory.

Definition 2 (a trajectory). A trajectory is composed of the GPS points arranged in the chronological order and is denoted by $TR = p_1, p_2, \ldots, p_n$.

Definition 3 (a stop point). A stop point $SP_i = (lat_i, lng_i, t^i_{start}, t^i_{end})$ denotes a location where a node arrives at the time t^i_{start} and departs at the time t^i_{end}.

3.1. Moving Speeds. Usually, the moving speed of a node is changed largely when its travel mode is switched [17]. The common travel modes include walking, riding bicycles, taking buses, driving cars, or taking the subway. As shown in Figure 2, suppose the node travels from the location A to the location B and the change point is the position where the node switches the travel mode from walking to taking the bus.

To detect the change point caused by the switch among different travel modes, we define the moving speed difference as $\triangle_{i+1} = |v_{i,i+1} - v_{i+1,i+2}|$, where $v_{i,i+1}$ represents the average speed on the line segment $p_i p_{i+1}$. If $\triangle_{i+1}$ is larger than a given threshold, then p_{i+1} will be retained as a change point.

Considering the fact that the speed changes are usually caused by the switches among different travel modes (Figure 3), we take the standard deviation of moving speeds as the speed threshold, since the standard deviation can reflect the speed fluctuations. The speed threshold v_{th} is expressed as

$$v_{th} = \sqrt{\frac{\sum_{i=1}^{n-1} (v_{i,i+1} - \overline{v})^2}{n-1}}, \quad (1)$$

where $\overline{v} = \sum_{i=1}^{n-1} v_{i,i+1}/(n-1)$; i.e., $\overline{v}$ denotes the average speed in a trajectory.

3.2. Stop Points. Another fact is that the nodes are possible to stay around some locations for a long period, e.g., waiting for

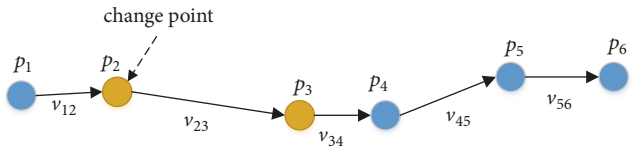


Figure 3: Speed change.

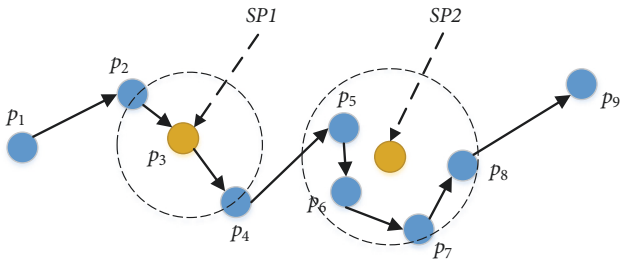


Figure 4: Two types of stop points.

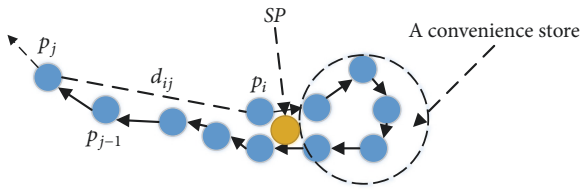


Figure 5: A false stop point.

the bus, having a dinner at a restaurant, resting in a leisure area, or shopping at a mall, and these locations can be taken as stop points. There are two types of stop points [18–20]: (*a*) the position-invariant stop points: such as the *SP1* in Figure 4, which is determined according to the preset staying period; (*b*) the position-offset stop points: the node moves around a position for a preset period, such as the *SP2* in Figure 4.

In previous works, the stay points are detected according to the distance threshold and the time threshold, i.e., the distance between an anchor point and its successors is judged whether it is larger than the distance threshold, and then the time span between the anchor point and the last successor is measured. If the time span is larger than the time threshold, a stay point is determined.

Notice that the stop point cannot be extracted when there is a round-trip path. As illustrated in Figure 5, a customer (node) visits a convenience store from p_i and then walks along the direction $\overrightarrow{p_i p_j}$. Assume the distance d_{ij} between p_i and p_j is larger than the distance threshold, while the time duration of the node traveling from p_i to p_{j-1} is also larger than the time threshold, and hence the average coordinate of all points from p_i to p_{j-1} can be denoted by the stop point, which actually leads to the extraction of a false stop point.

In addition, this process may fail to accurately detect the stop points due to the false position of the start point; as shown in Figure 6, if p_1 is taken as a start point mistakenly, then a false stop point is extracted.

Actually, the setting of the distance threshold will affect the detection accuracy of stop points. If a smaller distance

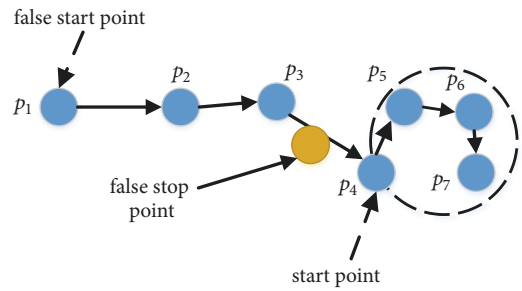


Figure 6: A false stop point.

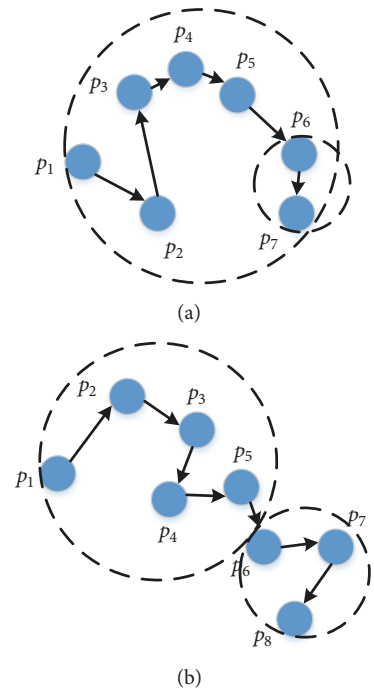


Figure 7: Two cases of distance threshold settings: (a) a smaller distance threshold and (b) a larger distance threshold.

threshold is set (e.g., 10m), a smaller portion of points located within the ranges of stay points is not detected, as shown in Figure 7(a), and p_6 and p_7 are not detected. If a large distance threshold is set (e.g., 100m), the points not falling into within range of stay point are also detected, as shown in Figure 7(b); p_6, p_7, and p_8 can be detected.

Furthermore, the time threshold is related to the number of stay points, and a smaller time threshold will result in more stop points; on the contrary, less stop points are detected if a larger time threshold is set.

To address the issues mentioned above, we observe that the speed of a node always becomes slow when it visits a location, and thus we propose a Stop Points Extraction Method based on the moving Speeds of nodes (SPEMS). In SPEMS, the stop points can be extracted through the following steps.

Step 1. Each pair of adjacent GPS points p_i, p_{i+1} is traversed, and the moving speed $v_{i,i+1}$ is calculated.

Input: $TR = \{p_1, p_2, \ldots, p_n\}, r, \lambda_v, \delta_t$.
Output: SP.

```
1:  i = 1.
2:  while i < n do
3:    j=i+1.
4:    Calculate v_{i,j}.
5:    if v_{ij} == 0 or v_{i,j} ≤ δ_v then
6:      L = Backward_Search(TR[ ], j, r).
7:      R = Forward_Search(TR[ ], j, r).
8:      △_t = p_R.t − p_L.t.
9:      if △_t ≥ δ_t then
10:       mid = ⌊(L + R)/2⌋.
11:       Add TR[mid] to SP.
12:       Remove points from p_L to p_R except p_mid.
13:       i = L.
14:     else
15:       i = i + 1.
16:     end if
17:   else
18:     i = i + 1.
19:   end if
20: end while
```

Algorithm 1: SPEMS pseudocode.

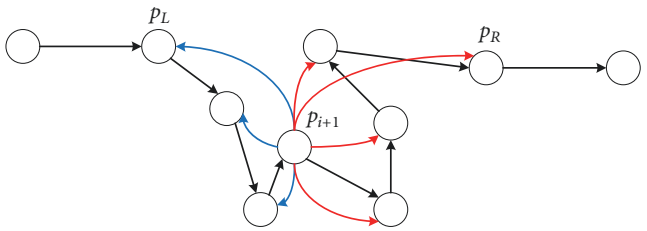


Figure 8: Forward search and backward search.

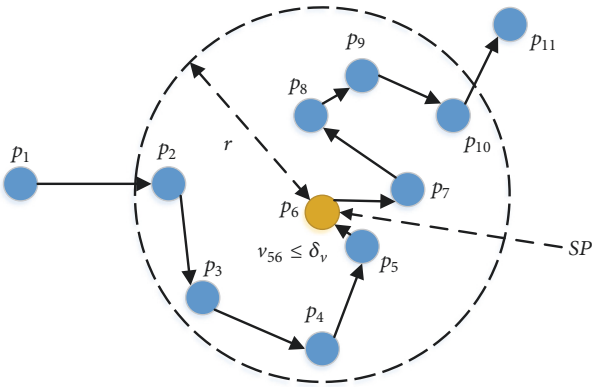


Figure 9: Stop point extraction.

Step 2. If $v_{i,i+1}$ is equal to 0 or $v_{i,i+1} \le \lambda_v$ (λ_v denotes a speed threshold), then a backward search with the center p_{i+1} is performed to extract a point p_L, such that the inequalities $dist(p_{i+1}, p_k) \le r$ and $dist(p_{i+1}, p_{L-1}) > r$ are satisfied, where r denotes a search radius threshold and $k = i, i-1, \ldots, L$.

Step 3. A forward search is performed to extract a point p_R, such that the inequalities $dist(p_{i+1}, p_k) \le r$ and $dist(p_{i+1}, p_{R+1}) > r$ are satisfied, where $k = i+2, i+3, \ldots, R$.

Step 4. If the time interval $\triangle_t(p_L, p_R) \ge \delta_t$ (δ_t denotes the time threshold), then we take the intermediate point $p_{\lfloor (L+R)/2 \rfloor}$ as a stop point. Obviously, we remove the points from p_L to p_R except $p_{\lfloor (L+R)/2 \rfloor}$, and this mechanism can avoid these points being repeatedly detected.

Figure 8 gives an example of forward search and backward search, where p_L is found by a backward search with the center p_{i+1} and p_R is found by a forward search.

An example is given in Figure 9, where we assume $v_{5,6} \ge v_{th}$ and hence we can find the points p_2 and p_{10} with the center p_6 through the backward search and the forward search, respectively. If $\triangle_t(p_2, p_{10}) \ge \delta_t$, we set $p_{\lfloor (2+10)/2 \rfloor}$ as the stop point; i.e., p_6 is a stop point.

The pseudocode of SPEMS is depicted in Algorithm 1.

3.3. Moving Directions. The changes of moving directions are concerned with the current travel modes; e.g., the moving direction will be changed more frequently when the nodes adopt the walking mode rather than taking vehicles.

Besides, the traditional DP algorithm cannot maintain the shapes of the original trajectories very well in some special cases, such as the case shown in Figure 10, where the point of the street corner is not taken as a feature point. The reason is that DP algorithm divides the trajectories on the basis of perpendicular Euclidean distance, and thus the points with large direction changes are easy to be ignored.

Input: $TR = \{p_1, p_2, \cdots, p_n\}$, **FP**, φ_{th}.
Output: **FP**.
1: s_{index}=0.
2: $e_{index} = len(TR) - 1$.
3: **if** $s_{index} < e_{index}$ **then**
4: $c_{index} = s_{index}$+1.
5: $\varphi_{max} = 0$.
6: $k_{index} = 0$.
7: **while** $c_{index} < e_{index}$ **do**
8: $\varphi = \theta_{max} \cdot dist(p_{index}, p_{index'})$.
9: **if** $\varphi \geq \varphi_{max}$ **then**
10: $\varphi_{max} = \varphi$.
11: $k_{index} = c_{index}$.
12: **end if**
13: $c_{index} = c_{index}$+1.
14: **end while**
15: **if** $\varphi_{max} \geq \varphi_{th}$ **then**
16: **FP**.$append(TR[k_{index}])$.
17: **DBDP**$(TR[s_{index} : k_{index}], \mathbf{FP}, \varphi_{th})$.
18: **DBDP**$(TR[k_{index} : e_{index}], \mathbf{FP}, \varphi_{th})$.
19: **end if**
20: **end if**

Algorithm 2: DPDPED pseudocode.

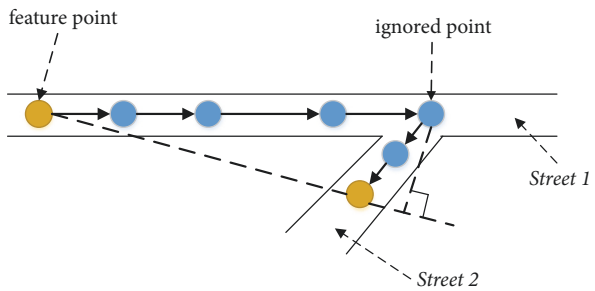


Figure 10: An ignored point in DP algorithm.

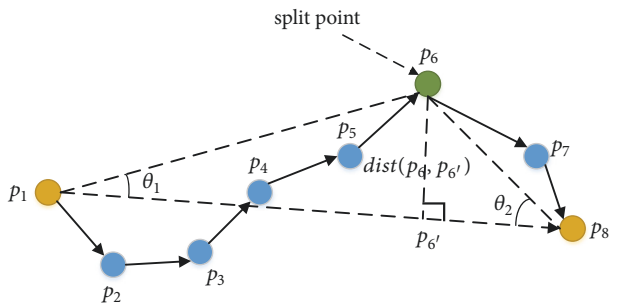


Figure 11: Illustration of DPDPED algorithm.

Therefore, we propose a DP algorithm based on Direction changes and Perpendicular Euclidean Distance (DPDPED). In DPDPED, the error measure is defined as

$$\varphi = \theta_{max} \cdot dist\left(p_k, p_{k'}\right), \quad i < k < j, \tag{2}$$

where $\theta_{max} = \max\{\theta_1, \theta_2\}$ and $0 \leq \theta_{max} \leq \pi$. θ_1 represents the angle between $\overrightarrow{p_i p_j}$ and $\overrightarrow{p_i p_k}$, θ_2 represents the angle between $\overrightarrow{p_i p_j}$ and $\overrightarrow{p_k p_j}$, where p_i and p_j are two adjacent feature points. p_k is a point on the line segment $\overrightarrow{p_i p_j}$, and $p_{k'}$ is the projection on $\overrightarrow{p_i p_j}$. Essentially, DPDPED combines the maximum angle and the perpendicular Euclidean distance into the index of error measure, and the points ignored by DP can be found through detecting the direction changes of nodes. In this mechanism, since the value of φ is very sensitive to the direction changes, and hence the points with large direction changes can be found easily. As illustrated in Figure 11, a split point p_6 divides the trajectory into two subtrajectories.

Algorithm 2 gives the pseudocode of DPDPED.

3.4. Trajectory Partition Algorithm. In TPMF, the change points are first extracted based on the speed changes, and then SPEMS algorithm is applied to extract the stop points. The extracted change points and stop points are treated as feature points. Finally, DPDPED is executed to simplify the subtrajectories between the adjacent feature points. Figure 12 illustrates our trajectory partition process, where two change points are found by the detection of speed changes in Figure 12(a), and then two stops points are extracted by our SPEMS in Figure 12(b), finally, a point with direction change largely is found by DPDPED in Figure 12(c). The pseudocode of TPMF is depicted in Algorithm 3.

According to the steps of TPMF, the time complexity is calculated as follows:

$$\begin{aligned} & n + n + \left(\left|\mathbf{FP_1}\right| - 1\right)\left[\frac{n - N_p - \left|\mathbf{FP_1}\right|}{\left|\mathbf{FP_1}\right|} + \frac{1}{2}\right]^2 \\ & = 2n + \left[\frac{\left|\mathbf{FP_1}\right| - 1}{4\left|\mathbf{FP_1}\right|^2} \times \left(2n - 2N_p - \left|\mathbf{FP_1}\right|\right)^2\right], \end{aligned} \tag{3}$$

Input: $TR = \{p_1, p_2, \ldots, p_n\}, r, \delta_t, \lambda_v$, the maximum angle threshold θ_{th}, the distance threshold d_{th}.
Output: Feature Point Set **FP**.
1: $i = 1, k = 0$, **FP** $= \Phi$.
2: **FP** $\longleftarrow$ (**FP** $\cup$ p_1), **FP** $\longleftarrow$ (**FP** $\cup$ p_n).
3: $v_{th} \longleftarrow$ Calculate standard deviation of speed in TR.
4: **while** $i \leq n - 2$ **do**
5: **if** $|v_{i,i+1} - v_{i+1,i+2}| \geq v_{th}$ **then**
6: **FP** $\longleftarrow$ (**FP** $\cup$ p_{i+1}).
7: **end if**
8: $i = i + 1$.
9: **end while**
10: **SP** $= SPEMS(TR, r, \delta_v, \delta_t)$.
11: **FP** $\longleftarrow$ (**FP** $\cup$ **SP**).
12: Sort the points of **FP** in a chronological order.
13: $\mathbf{FP_1} \longleftarrow$ (**FP**).
14: **Index** $\longleftarrow$ find the indexes of feature points in **FP1** in TR.
15: $\varphi_{th} = \theta_{th} \cdot d_{th}$.
16: **while** $k <$ *len*(**Index**) $- 1$ **do**
17: $DPDPED(TR[\mathbf{Index}[k] : \mathbf{Index}[k+1]], \mathbf{FP}, \varphi_{th})$.
18: $k = k + 1$.
19: **end while**
20: Remove duplicate points from **FP**.
21: Sort points in **FP** in a chronological order.

Algorithm 3: TPMF pseudocode.

where N_p is the number of removed points within the ranges of stop points and the expression of N_p is given by

$$N_p = \sum_{i=1}^{|\mathbf{SP}|} Num\left(SP_i\right), \quad SP_i \in \mathbf{SP}, \tag{4}$$

where SP_i denotes the i-th stop point in **SP** and $Num(SP_i)$ denotes the number of points around the stay point SP_i. $(\lfloor (n - N_p - |\mathbf{FP_1}|)/|\mathbf{FP_1}| + 1/2 \rfloor)$ denotes the average number of points between adjacent feature points. Therefore, the time complexity of TPMF is written as $\mathbf{O}(\lfloor (|\mathbf{FP_1}| - 1)/4|\mathbf{FP_1}|^2 \times (2n - 2N_p - |\mathbf{FP_1}|)^2 \rfloor)$. When n is large enough, we have

$$n < \left\lfloor \frac{|\mathbf{FP_1}| - 1}{4\left|\mathbf{FP_1}\right|^2} \times \left(2n - 2N_p - \left|\mathbf{FP_1}\right|\right)^2 \right\rfloor < n^2 \tag{5}$$

3.5. Trajectory Partition Error. Two metrics have been applied to measure the performance of trajectories partition methods: (*a*) simplification rate refers to the ratio of removed points to original points and (*b*) running time represents the execution time of trajectory partition method to measure the time complexity. Besides, the error between the simplified trajectories and the original trajectories should also be measured, and we define the metric of trajectory partition error, as shown in Figure 13, p_i and p_j are two adjacent feature points, and the average perpendicular Euclidean distance and the average angle are used to evaluate the error of trajectory partition results.

The expression of average perpendicular Euclidean distance is given by

$$avg_p = \frac{\sum_{i=\mathbf{F}_I[\alpha], j=\mathbf{F}_I[\alpha+1]} \sum_{k=i+1}^{j-1} dist\left(p_k, p_{k'}\right)}{n - s}, \quad \alpha = 0, 1, \ldots, \left|\mathbf{F}_I\right| - 1, \tag{6}$$

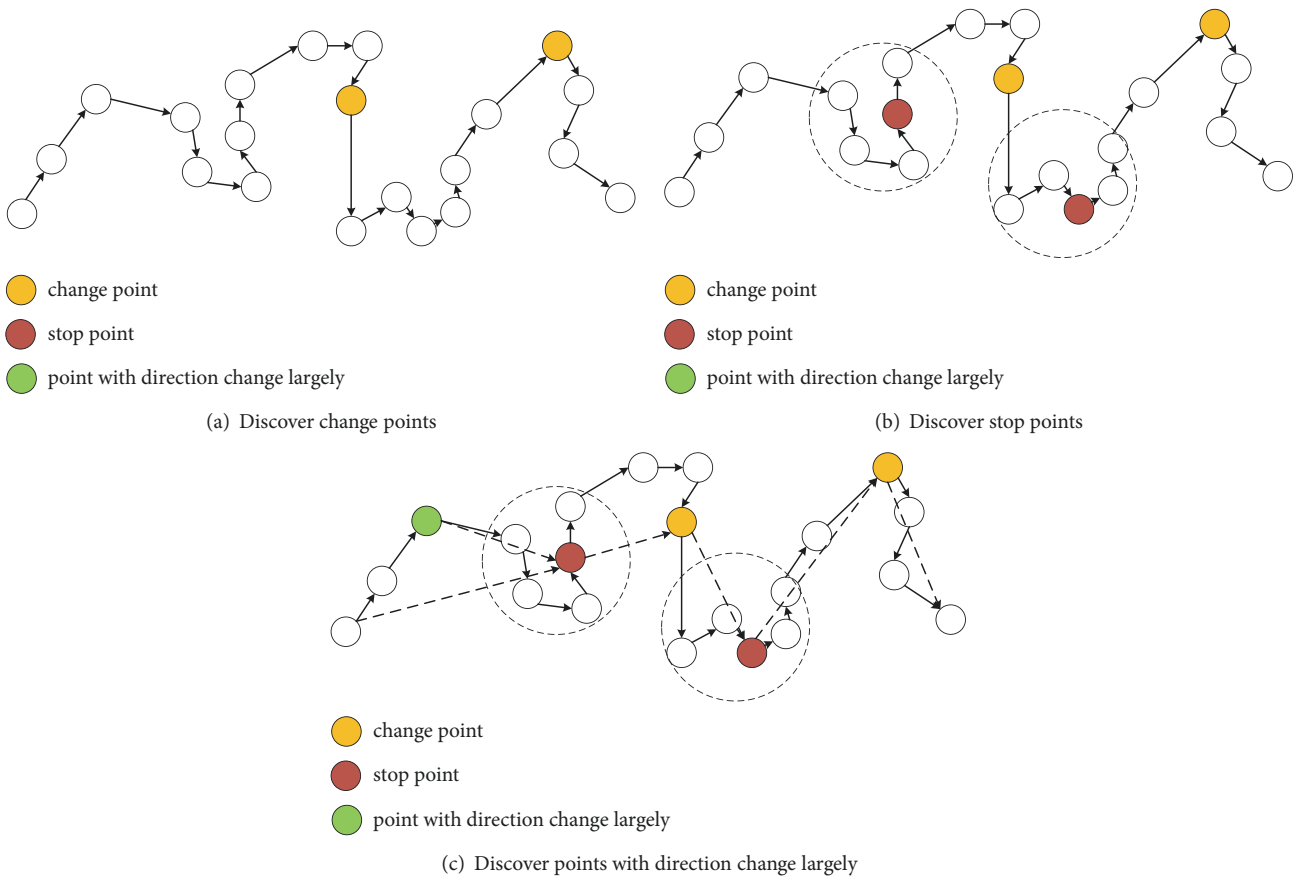
where $\mathbf{F}_I$ denotes the subscript set of feature points, $dist(p_k, p_{k'})$ represents the perpendicular Euclidean distance between p_k and $p_{k'}$, and s denotes the number of points in **FP**.

The expression of average angle is written as

$$avg_a = \frac{\sum_{i=\mathbf{F}_I[\alpha],\ j=\mathbf{F}_I[\alpha+1]} \left(\left(\max \theta_{i,j} + \max \theta_{j,i}\right)/2\right)}{s - 1}, \quad \alpha = 0, 1, \ldots, \left|\mathbf{F}_I\right| - 1, \tag{7}$$

where $\theta_{i,j} = \{\theta_{i,i+1}, \theta_{i,i+2}, \ldots, \theta_{i,k}, \ldots, \theta_{i,j-1}\}$ represents the set of angles between $\overrightarrow{p_i p_k}$ and $\overrightarrow{p_i p_j}$ and $\theta_{j,i} = \{\theta_{j,j-1}, \theta_{j,j-2}, \ldots, \theta_{j,k}, \ldots, \theta_{j,i+1}\}$ represents the set of angles between $\overrightarrow{p_j p_k}$ and $\overrightarrow{p_j p_i}$.

Furthermore, the Min-Max standardization is used to standardize avg_p and avg_a, respectively. Finally, we define the trajectory partition error $\varepsilon = w_1 \cdot avg_p^* + w_2 \cdot avg_a^*$, where w_1 and w_2 are the weights reflecting the impacts of average perpendicular Euclidean distance and average angle on the



(a) Discover change points

(b) Discover stop points

(c) Discover points with direction change largely

FIGURE 12: Trajectory partition process.

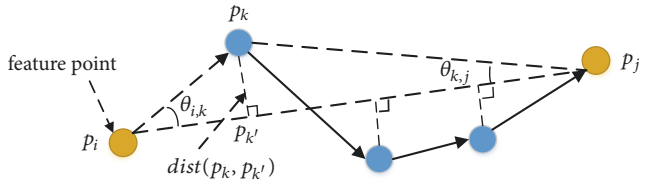


FIGURE 13: Trajectory partition error.

trajectory partition. Then, the expressions of avg_p^* are written as

$$avg_p^* = \frac{avg_p - min_p}{max_p - min_p}, \tag{8}$$

where min_p denotes the minimum value of all the perpendicular Euclidean distances from the points between adjacent feature points to the simplified line segment. Likewise, max_p represents the maximum value.

The expressions of avg_a^* are written as

$$avg_a^* = \frac{avg_a - min_a}{max_a - min_a}, \tag{9}$$

where min_a denotes the minimum value of all average angles between adjacent feature points and max_a denotes the maximum value.

4. Simulations

In this section, we present an extensive simulation study of our TPMF. We conduct several simulations to evaluate TPMF on the *Geolife* dataset. All simulations are run on a PC equipped with Windows 7, 3.20GHz CPU and 4 GB memory. The trajectory partition methods are realized by Python language.

4.1. Dataset. GPS trajectory dataset is cited from (Microsoft Research Asia) *Geolife* [21] project, which collects the trajectories of 182 users (nodes) during five years (from April 2007 to August 2012). These trajectories are recorded by different GPS loggers and GPS phones and have a variety of sampling rates. 91.5 percents of the trajectories are logged in a dense representation, e.g., every 1~5 seconds or every 5~10 meters per point.

4.2. Parameter Settings. The trajectory data used in our simulations includes 171 trajectories, each of which contains tracks of one node every day. Firstly, we select six trajectories from the 171 trajectories to determine the parameter values. The details of each trajectory are shown in Table 1.

To observe the impacts of δ_t and r on the number of stop points, four trajectories are selected for this simulation. Figure 14 shows the number of stop points under different search radius and time threshold.

Table 1: Trajectory details.

TraID	Date	Start	End	Raw Points
1	2008-10-23	02:53:04	11:11:12	908
5	2008-10-28	00:38:26	05:03:42	1477
9	2008-11-03	23:21:53	03:31:08	2231
11	2008-11-11	00:17:04	02:35:54	681
49	2009-04-05	05:19:38	14:00:18	4004
161	2009-06-28	00:52:29	13:45:56	5848

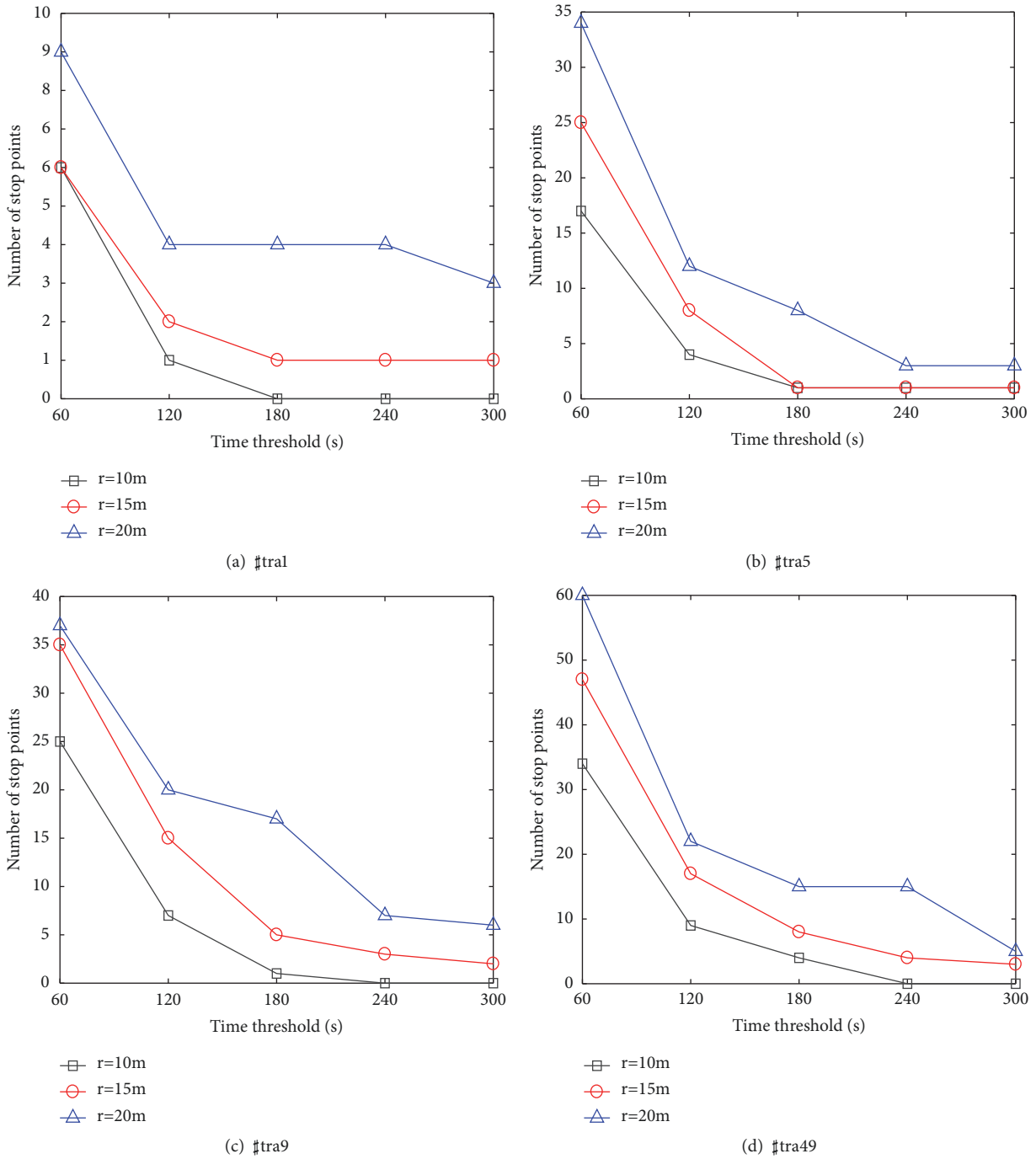


Figure 14: Number of stop points vs. time threshold and search radius threshold.

Table 2: Simulation parameters.

Parameters	Description	Values
δ_t	Time threshold	240s
r	Search radius threshold	20m
λ_v	Speed threshold	0.6m/s
d_{th}	Distance threshold	10m
θ_{th}	Maximum angle threshold	$\pi/6$
w_1	Weight of avg_p^*	1/2
w_2	Weight of avg_a^*	1/2

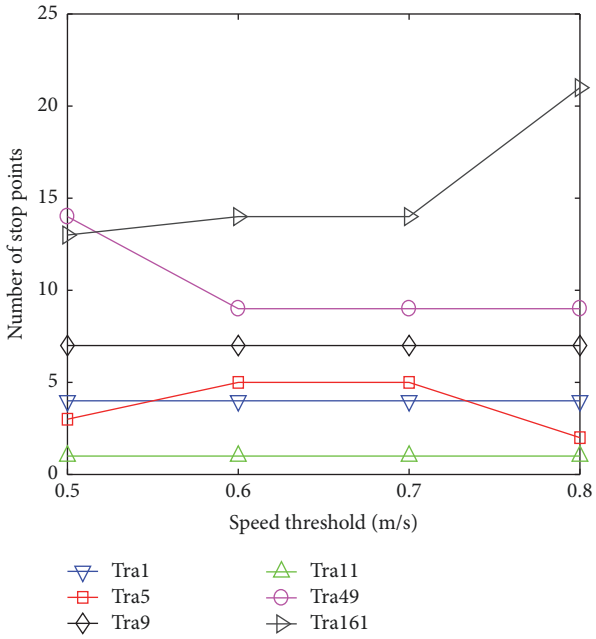


Figure 15: Number of stop points vs. moving speed threshold.

From Figures 14(a)–14(d), we can observe that the curves decrease rapidly when δ_t increases from 60 s to 120 s, which is attributed to the fact in TPMF more points are treated as the stop points when a smaller time threshold δ_t is set. When $\delta_t \geq 120$ s, the curves descend slowly, especially when 180 s$\leq \delta_t \leq$300 s the curves remain almost stable, which indicates that the fluctuation of the number of stop points is very slight, and the reason is that the range of 180 s$\leq \delta_t \leq$300 s is close to the duration that the node stays at the stop points, and thus the value of δ_t should be selected from the interval [180, 300] s. In addition, when r=10 m or r=15 m, the number of extracted stop points is not large enough, compared with the number of original points, and thus we set r=20 m. In Figures 14(a) and 14(d), the number of stop points does not change obviously when 180 s$\leq \delta_t \leq$240 s. Similarly, Figures 14(b) and 14(c) also illustrate that the number of stop points is not changed when 240 s$\leq \delta_t \leq$300 s, and thus we set δ_t=240 s.

As shown in Figure 15, the number of stop points remains unchanged when 0.6 m/s$\leq \lambda_v \leq$0.7 m/s, which is attributed to the fact that the speed falling into the interval of 0.6 m/s$\leq \lambda_v \leq$0.7 m/s is very close to the moving speed of the node, and we can set λ_v = 0.6 m/s. Since the trajectory partition error increases (or decreases) as d_{th} or θ_{th} increases (or decreases), there are no suitable values can be obtained. Thus, we set d_{th} and θ_{th} smaller in our simulations. Without loss of generality, we assume the impacts of average perpendicular Euclidean distance and average angle be equivalent, and each of w_1 and w_2 is set to 0.5. Actually, w_1 and w_2 are adjustable weights, and the values of w_1 and w_2 can be easily adjusted for different applications or different trajectory datasets. The main simulation parameters are provided in Table 2.

4.3. Comparisons of Trajectory Partition Methods. Seven algorithms are compared in this simulation. Our proposed algorithm TPMF is compared with other six algorithms (DP, MDL, SQUISH, SQUISH-E, BQS, and OPERB) in terms of simplification rate, running time, and trajectory partition error. The parameter settings of other algorithms as given in Table 3, where PED represents the Perpendicular Euclidean Distance and SED represents the Synchronized Euclidean Distance. The table term "yes/no" indicates whether the algorithm uses PED or SED. We first partition all the trajectories of one node, and the simulation results are reported in Figure 16.

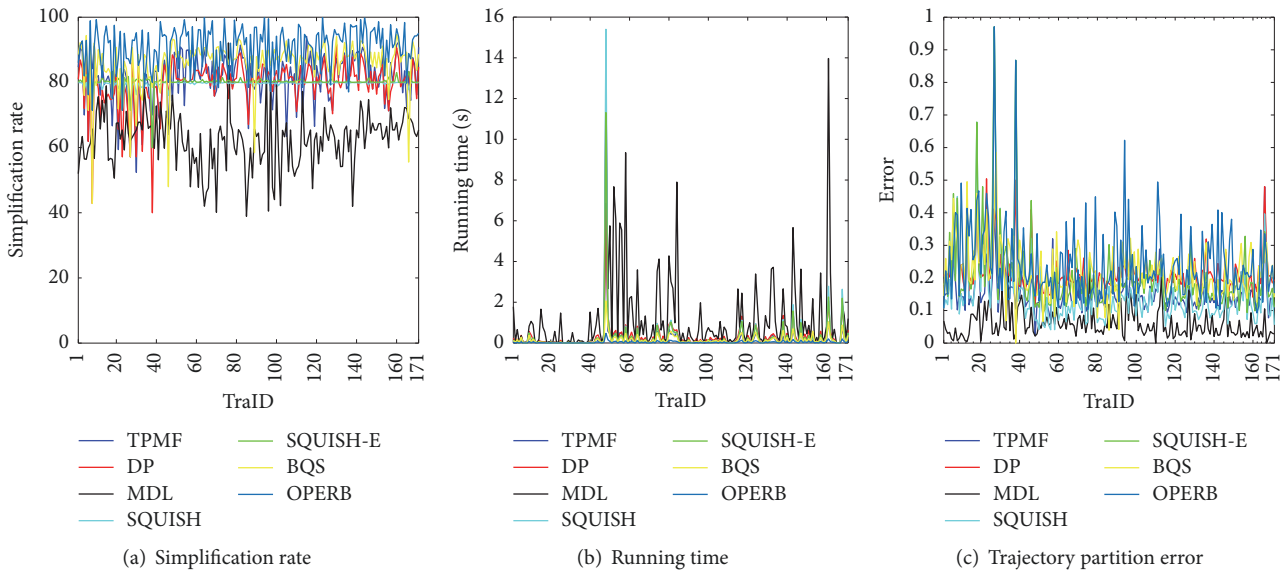
As shown in Figure 16(a), with regard to the simplification rate, OPERB outperforms other algorithms and the curves of TPMF and BQS higher than those of DP, MDL, SQUISH, and SQUISH-E. Particularly, MDL always achieves the lowest simplification rates. As illustrated in Figure 16(b), the running time of MDL is much longer than other algorithms. In Figure 16(c), MDL outperforms other algorithms in terms of trajectory partition error.

To observe the impacts of the number of trajectories, we select 10 nodes from the dataset, and the trajectory details are shown in Table 4. The simulation results are given in Figure 17.

In Figure 17(a), with regard to the average simplification rate, OPERB is generally higher than other algorithms, and the simplification rate of OPERB is about 90%. the plots of TPMF and BQS are close to each other, and their simplification rates fall into the interval from 85% to 90%. The plot of DP is slightly lower than those of TPMF and BQS. Especially, SQUISH and SQUISH-E remain stable due to the constraint of a target simplification rate 80%, and MDL obtains the lowest simplification rates among all algorithms.

Figure 17(b) illustrates that OPERB executes faster than other algorithms due to its one-pass scanning mechanism, while the running time of MDL is much longer than others. Besides, TPMF consumes less running time than those of DP, MDL, SQUISH, SQUISH-E, and BQS. This is attributed to the fact that TPMF partitions the trajectories according to the extracted feature points (change points and stop points), and hence the number of computations is significantly reduced.

In Figure 17(c), the average error of TPMF is always smaller than those of DP, BQS, SQUISH-E and OPERB and is close to that of SQUISH. MDL achieves the lowest average error, and this is because MDL partitions the trajectories approximately, and it retains the most points and preserves the trajectory shapes as much as possible, which produces a



(a) Simplification rate (b) Running time (c) Trajectory partition error

Figure 16: Comparisons of Different Trajectory Partition Algorithms (one node).

Table 3: Parameter settings of other algorithms.

Algorithms	PED	SED	λ	μ
DP	yes	no	-	10m
MDL	no	no	-	-
SQUISH	no	yes	5	-
SQUISH-E	no	yes	5	10m
BQS	yes	no	-	10m
OPERB	yes	no	-	10m

Table 4: Trajectory detail of nodes.

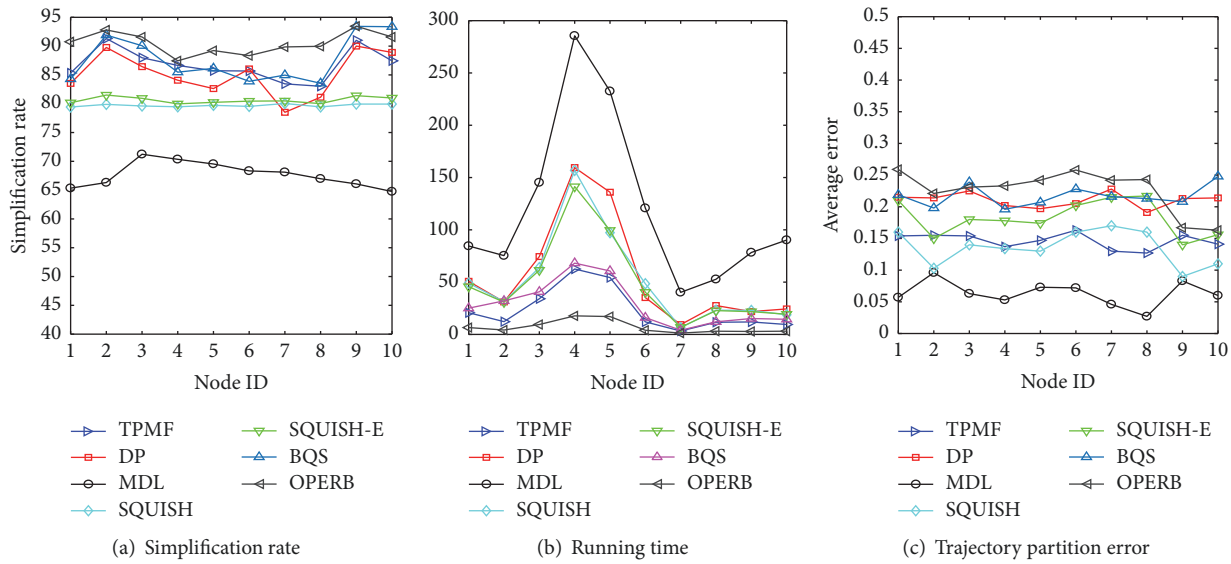
Node ID	Original trajectories	Original points	Time periods
1	171	173870	2008-10-23~2009-07-05
2	71	108607	2008-10-23~2008-12-14
3	175	248215	2008-10-23~2009-03-22
4	322	485226	2008-10-23~2009-07-05
5	395	439397	2008-10-23~2009-07-29
6	86	109046	2008-10-24~2009-03-19
7	28	31830	2008-10-23~2008-12-11
8	54	85531	2008-10-25~2008-12-15
9	34	77908	2008-10-24~2008-12-12
10	49	84615	2008-10-23~2008-12-14

smaller trajectory partition error along with a lower simplification rate. Notice that OPERB obtains the largest average error along with the highest simplification rate as depicted in Figure 17(a).

Therefore, TPMF makes a preferable tradeoff between the simplification rate and the trajectory partition error. TPMF can reserve the most valuable feature points, which makes the redundant points be removed.

5. Conclusions

The trajectory partition is a primary procedure of trajectory data mining. The amount of movement trajectories is typically large, and the shapes of trajectories appear diversely in different applications. In this paper, a trajectory partition method based on combined movement features (moving speeds, stop points, and moving directions) is proposed.



(a) Simplification rate (b) Running time (c) Trajectory partition error

Figure 17: Comparisons of Different Trajectory Partition Algorithms (ten nodes).

The outstanding advantage of our proposed method is that it takes into account the movement behaviors of nodes comprehensively, which can maintain the shape skeletons and movement features while massive redundant points are removed.

Future research will focus on investigating a self-adaptive solution of setting the method parameters in TPMF, e.g., the values of parameters r, δ_t, and λ_v. In addition, a trajectory mining algorithm based on TPMF will be investigated to obtain the movement modes of nodes as well.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that they have no conflicts of interest.

Acknowledgments

This research is supported by National Natural Science Foundation of China (Grant Nos. 61872191, 41571389, and 61801215); Postdoctoral Science Foundation of China (Grant Nos. 2014M560379, 2015T80484); Natural Science Foundation of Jiangsu Province (Grant No. BK20160812).

References

[1] Y. Takeuchi and M. Sugimoto, "CityVoyager: an outdoor recommendation system based on user location history," in *Proceedings of the 3rd International Conference on Ubiquitous Intelligence and Computing*, pp. 625–636, 2006.

[2] L. Chen, M. Lv, and G. Chen, "A system for destination and future route prediction based on trajectory mining," *Pervasive and Mobile Computing*, vol. 6, no. 6, pp. 657–676, 2010.

[3] Y. Yu, J. Kim, K. Shin, and G. S. Jo, "Recommendation system using location-based ontology on wireless internet: An example of collective intelligence by using 'mashup' applications," *Expert Systems with Applications*, vol. 36, no. 9, pp. 11675–11681, 2009.

[4] W. C. Lee and J. Krumm, *Trajectory Preprocessing, Computing with Spatial Trajectories*, Springer, 2011.

[5] D. Douglas and T. Peucker, "Algorithms for the reduction of the number of points required to represent a line or its caricature, cartographica," *The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[6] F. Zhang and X. M. Zhang, "A spatiotemporal compression algorithm for gps trajectory data," *Computer and Communications*, vol. 31, p. 6, 2013.

[7] J. Muckell, J. Hwang, V. Patil, C. T. Lawson, F. Ping, and S. S. Ravi, "SQUISH: an online approach for GPS trajectory compression," in *Proceedings of the 2nd International Conference on Computing for Geospatial Research and Applications*, vol. 13, 2011.

[8] J. Muckell, P. W. Olsen Jr., J.-H. Hwang, C. T. Lawson, and S. S. Ravi, "Compression of trajectory data: a comprehensive evaluation and new approach," *GeoInformatica*, vol. 18, no. 3, pp. 435–460, 2014.

[9] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, and R. Jurdak, "Bounded Quadrant System: Error-bounded trajectory compression on the go," in *Proceedings of the 2015 31st IEEE International Conference on Data Engineering, ICDE 2015*, pp. 987–998, Republic of Korea, April 2015.

[10] X. Lin, S. Ma, H. Zhang, T. Wo, and J. Huai, "One-pass error bounded trajectory simplification," *Proceedings of the VLDB Endowment*, vol. 10, no. 7, pp. 841–852, 2017.

[11] Y. Zheng and X. Zhou, Eds., *Location-Based Social Networks: Users, Computing with Spatial Trajectories*, Springer, New York, NY, USA, 2011.

[12] K. Richter, F. Schmid, and P. Laube, "Semantic trajectory compression: Representing urban movement in a nutshell," *Journal of Spatial Information Science*, pp. 3–30, 2012.

[13] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: a partition-and-group framework," in *Proceedings of the ACM*

SIGMOD International Conference on Management of Data (SIGMOD '07), pp. 593–604, ACM, Beijing, China, June 2007.

[14] H. Yuan, Y. Qian, R. Yang, and M. Ren, "Human mobility discovering and movement intention detection with GPS trajectories," *Decision Support Systems*, vol. 63, pp. 39–51, 2014.

[15] Z. Fu, Z. Tian, Y. Xu, and K. Zhou, "Mining frequent route patterns based on personal trajectory abstraction," *IEEE Access*, vol. 5, pp. 11352–11363, 2017.

[16] X. Gao and F. Yu, "Trajectory clustering using a new distance based on minimum convex hull," in *Proceedings of the 9th International Conference on Soft Computing and Intelligent Systems*, pp. 1–6, 2017.

[17] Y. Zheng, Y. Chen, Q. Li, X. Xie, and W.-Y. Ma, "Understanding transportation modes based on GPS data for web applications," *ACM Transactions on the Web (TWEB)*, vol. 4, no. 1, pp. 1–36, 2010.

[18] Q. Li, Y. Zheng, X. Xie, Y. Chen, W. Liu, and W. Y. Ma, "Mining user similarity based on location history," in *Proceedings of the 16th ACM Sigspatial International Conference on Advances in Geographic Information Systems*, p. 34, ACM, New York, NY, USA, 2008.

[19] X. Chen, B. Zhao, D. Shi, and F. Liu, "Mining individual mobility patterns based on location history," in *Proceedings of the 1st IEEE International Conference on Data Science in Cyberspace, DSC 2016*, pp. 252–259, China, June 2016.

[20] Y. Zheng, "Trajectory Data mining: an overview," *ACM Translations on Intelligent System and Technology*, vol. 6, no. 3, Article ID 29, 2015.

[21] Z. Yu, X. Xing, and W.-Y. Ma, "Geolife: a collaborative social networking service among user, location and trajectory," *IEEE Data Engineering Bulletin*, vol. 33, no. 2, pp. 32–40, 2010.