Streaming Trajectory Segmentation Based on Stay-Point Detection

Yangyang Sun[1,2], Fei Meng[1,2], Ruiyuan Li[1,2*✉],
Yongxin Tang[3,2], Chao Chen[1*✉], and Jiang Zhong[1]

[1] Chongqing University, Chongqing 400010, China
{sunyangyang, mengfei, ruiyuan.li, cschaochen, zhongjiang}@cqu.edu.cn
[2] Start (Spatio-Temporal Art) Lab, Chongqing 400010, China
[3] Shaoyang University, Shaoyang 422000, China
tangyongxin@kangry.net

Abstract. Trajectory segmentation is essential for various location-based applications. Most of the existing trajectory segmentation algorithms are designed for offline data, thus cannot segment trajectory streams efficiently. In this paper, we propose a novel and efficient streaming trajectory segmentation algorithm based on stay point detection. Our algorithm can dynamically update in real time based on input points, eliminating the need for scanning the entire trajectory. To enhance its suitability for streaming scenarios, we introduce a well-designed grid index and identify three areas based on it for calculation pruning. We conduct extensive experiments using three diverse datasets from different domains (ranging from taxis to trucks to pedestrians), which verifies the efficiency of our method.

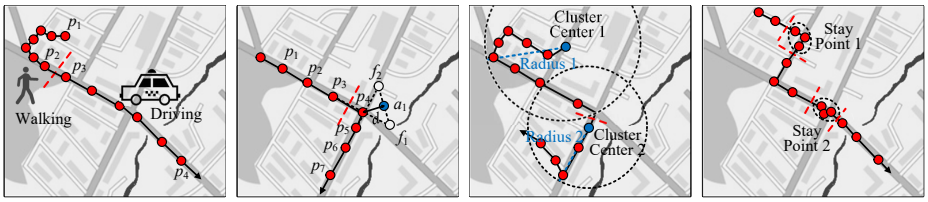
Keywords: trajectory data mining · streaming processing.

1 Introduction

With the increasing popularity of mobile devices such as smartphones and positioning systems, a large number of trajectory data streams are being generated. The analysis of these trajectory streams can yield valuable insights and contribute to the advancement of modern applications, such as traffic congestion avoidance [1], and identification of high-demand areas [6].

Trajectory segmentation, which divides a long trajectory into multiple shorter ones [14, 17], is a crucial and fundamental preprocessing technique. We usually organize a trajectory consisting of the GPS (Global Position System) records produced by the same moving object. Since the GPS records are constantly generated, a complete trajectory is in fact an infinite stream that extends into the future. Segmenting streaming trajectories allows for the division of a trajectory stream into shorter sub-trajectories, playing a crucial role in stream trajectory processing. First, for trajectory management systems like TrajMesa [7,

* Ruiyuan Li and Chao Chen are the corresponding authors of this paper.



(a) Feature-Based (b) Interpolation-Based (c) Cluster-Based (d) Stay Point-Based

Fig. 1: Categories of Trajectory Segmentation.

8], TraSS [4] and TMan [5] that primarily store a historical sub-trajectory as a record, it is often not possible to direct query the most recent trajectories. Streaming trajectory segmentation can offer a solution by providing these systems with the latest sub-trajectories. Second, segmenting a lengthy trajectory stream into multiple shorter ones can significantly decrease memory usage. This optimization also enhances the efficiency of trajectory mining tasks, whose computational complexity is influenced by the number of GPS points in a trajectory, such as map-matching [11, 17] and path querying [9, 10].

Existing trajectory segmentation algorithms can be classified into four categories: feature-based segmentation [12, 16], interpolation-based segmentation [2, 3], cluster-based segmentation [13], and stay point-based segmentation [14, 15]. Feature-based methods typically employ feature variation thresholds to determine whether to segment a trajectory or not. These features could be time interval, velocity, acceleration, jerk, direction, travel mode, and so on. As shown in Fig. 1(a), the moving object has two travel modes, so its trajectory is segmented into two sub-trajectories. However, it is sometimes hard to obtain these features, making the methods of this kind inapplicable. Interpolation-based methods first predict a GPS point through interpolation and then compare the distance between the predicted point and the actual one. If the distance is big enough, it performs a segmentation. As illustrated in Fig. 1(b), SWS [3] first predicts the point f_1 using p_1, p_2 and p_3, and predicts f_2 using p_5, p_6 and p_7. After that, it calculates the center point a_1 of f_1 and f_2, and measures the distance between p_4 and a_1. Since the methods of this type utilize the points both before and after the predicted point, they introduce much latency. Cluster-based methods regulate the segmentation by clusters. For example, as shown in Fig. 1(c), the work [13] first calculates the distance between the first point and each following point in a sub-trajectory, and then computes the ratio of point number to the area of the circle with the first point as its center and the maximum distance as its radius. If the ratio is greater than a threshold, it forms a sub-trajectory. However, this method [13] usually leads to meaningless segmentations. To address the issue, stay point-based methods [15] first extract the stay points [14] in a trajectory, and then segment the trajectory by the stay points, as shown in Fig. 1(d). Stay points usually represent meaningful activities, such as pick-ups or drop-offs of a taxi. However, stay point detection adopted by the work [15] is time-consuming, which is not suitable for streaming scenarios.

It is challenging to segment trajectories in streaming scenarios. *Challenge 1*, how to use partial data for trajectory segmentation? In streaming environments, the GPS points are continuously coming. Moreover, due to the limitation of memory, the historical trajectory cannot be stored permanently. *Challenge 2*, how to use only the limited features for meaningful trajectory segmentation? Since most trajectories only contain the latitude, longitude and timestamp information, using only this information can extend the application scenarios. *Challenge 3*, how to design an efficient computational framework to support trajectory segmentation? Since the GPS points in a trajectory are continuously generated at an extremely high rate, it is of vital importance to handle them efficiently.

To address the aforementioned challenges, we propose a Streaming Trajectory sEgmentation framework based on stay Points (i.e., STEP). Overall, the main contributions of this paper are as follows:

• We propose a streaming trajectory segmentation framework based on stay point detection, i.e., STEP. STEP can rely only on recent data and a few features to efficiently divide trajectories into meaningful trajectory segments.

• We propose a well-designed grid index and three areas to prune unnecessary distance calculations effectively, accelerating the process of trajectory segmentation in streaming environments.

• We conduct extensive experiments on three different types of datasets (ranging from taxis to trucks to pedestrians), verifying the superiority of our proposed method in terms of latency and throughput.

2 Perliminaries

Definition 1 ***GPS Point.*** *A GPS point $p = (lat, lng, t)$ consists of a latitude lat, a longitude lng, and a timestamp t. It represents that the moving object is located at the geographic coordinates (lat, lng) at time t.*

Definition 2 ***Trajectory.*** *A trajectory $tr = \{p_1 \rightarrow p_2 \rightarrow ...\}$ is a sequence of GPS points generated by the same moving object and ordered by their timestamps, i.e., for all $i \geq 1$ we have $p_i.t < p_{i+1}.t$.*

Definition 3 ***Sub-trajectory.*** *A sub-trajectory $str = \{p_m \rightarrow p_{m+1} \rightarrow ... \rightarrow p_n\}$ is a continuous subsequence of the trajectory tr.*

Definition 4 ***Stay Point.*** *Given two user-specified distance parameter D and time parameter T, a stay point $SP = \{p_a \rightarrow p_{a+1} \rightarrow ... \rightarrow p_b\}$ is a sub-trajectory, satisfying the following two constraints: (1) time constraint, i.e., $p_b.t - p_a.t \geq T$; (2) distance constraint, i.e., $d(p_a, p_i) \leq D$ but $d(p_a, p_{b+1}) > D$ for all $a < i \leq b$, where $d(*, *)$ is the distance of two GPS points.*

Definition 5 ***Stay Point-Based Trajectory Segmentation.*** *Given a trajectory tr, distance constraint D and time constraint T, we break tr into the minimum k sub-trajectories $\{str_1, str_2, ..., str_k\}$, such that: (1) $str_i \cap str_j = \emptyset$ for any $i \neq j$, i.e., str_i and str_j do not share any GPS points; (2) $\cup_{i=1}^{k} str_i = tr$; and (3) for any $1 \leq i \leq k$, the GPS points in str_i do not belong to any stay point, or all GPS points in str_i belong to at least one stay point.*

As shown in Fig. 2, the trajectory tr can be segmented into three sub-trajectories $\{str_1, str_2, str_3\}$. The GPS points in str_1 and str_2 do not belong to any stay point, while the GPS points in str_2 belong to SP_1 or SP_2. Note that in streaming scenarios, new GPS points are constantly generated, making the last sub-trajectory str_3 unstable (i.e., the GPS points of str_3 may increase). We define the status of an unstable sub-trajectory as **open**, and the status of a stable sub-trajectory as **closed**.

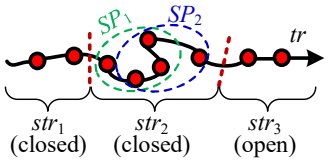


Fig. 2: Stay Point-Based Trajectory Segmentation.

3 Framework

The framework of STEP, as illustrated in Fig. 3, comprises three modules: *Indexing*, *Stay Point Detection* and *Trajectory Segmentation*.

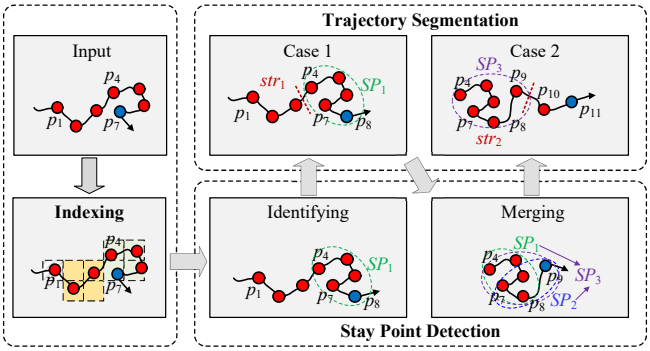


Fig. 3: Framework of STEP.

Indexing. In this module, we build a grid index for the input trajectories. With the grid index, we can avoid unnecessary expensive distance calculations, and accelerate the process of stay point detection (detailed in Section 4).

Stay Point Detection. In this module, we detect the stay points with two operations: 1) identifying, which checks if the newly arrived GPS point and several of its previous GPS points constitute a stay point, and 2) merging, which merges two stay points into a big one if they share some GPS points [1] (detailed in Section 5).

Trajectory Segmentation. In this module, we perform segmentation operators in two cases. Case 1, if the newly arrived GPS point belongs to a stay point, we segment the GPS points that do not belong to any stay point (if any), view them as a closed sub-trajectory (e.g., str_1 in Fig. 3) and flush them out of the memory. Case 2, if the newly arrived GPS point will not belong to the previous stay point (if any), we regard the previous stay point as a closed sub-trajectory (e.g., str_2 in Fig. 3) and flush it out of the memory (detailed in Section 6).

[1] The merged stay point may not satisfy the distance constraint, but it does not affect the correctness of segmentation. On the contrary, it can simplify the description.

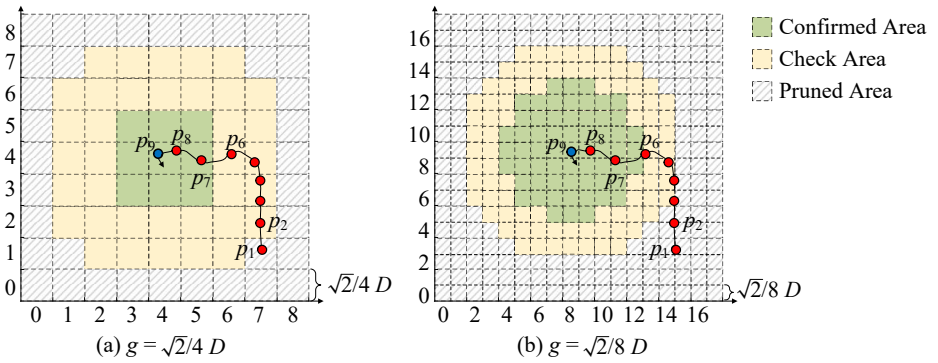


Fig. 4: Examples of Grid Index.

4 Indexing

Once a new GPS point is coming, we need to trigger a stay point detection process. The naive stay point detection method first finds the latest GPS point p_{i-1} that does not satisfy the distance constraint and then checks if the time constraint between p_c and p_i is satisfied. If so, the points from p_i to p_c form a stay point. Otherwise, they do not. The algorithm contains many distance calculations, which are relatively time-consuming. To this end, we propose a light-weight grid index to speed up stay point detection.

The grid index partitions the geographic space into equal-sized and disjoint grids, where the grid size g is carefully designed according to the user-specified parameter D (see below). Each grid is bounded with a coordinate (x, y). A GPS point $p_i = (lat_i, lng_i, t_i)$ is located in the grid (x_i, y_i) that

$$x_i = \lfloor (lat_i - lat_{min})/g \rfloor, y_i = \lfloor (lng_i - lng_{min})/g \rfloor \tag{1}$$

where lat_{min} and lng_{min} are the minimum latitude and longitude of interest, respectively. It is fast to calculate the located grid of a GPS point. In what follows, we use $(p_i.x_i, p_i.y_i)$ or $(p_i.x, p_i.y)$ to represent the located grid of p_i. Given the current point p_c and a previous point p_i, we define Δx and Δy as:

$$\Delta x = |p_i.x - p_c.x|, \Delta y = |p_i.y - p_c.y| \tag{2}$$

It is of great importance to determine the grid size g. Intuitively, g should be as small as possible, thus achieving a fine-grained location representation. As shown in Fig. 4(a), when g is set $\frac{\sqrt{2}}{4}D$, the grids can be clearly grouped into three areas: confirmed area (in green), check area (in yellow) and pruned area (in shadow). Here, p_9 is the current GPS point, located in the center grid of the three areas. The distance between p_9 and any point located in the confirmed area (e.g., p_8 or p_7) must be less than D, while the distance between p_9 and any points located in the pruned area (e.g., p_1) must be greater than D. The GPS points in the check area (e.g., p_6) need to be further checked. If the grid size g is set $\frac{\sqrt{2}}{8}D$, as shown in Fig. 4(b), we can get a larger confirmed area

Algorithm 1: Indexed Stay Point Detection

Input : Current GPS point p_c, previous GPS points in the cache, grid index, distance parameter D, time parameter T
Output: Stay point (if any)

```
i ← c − 1;
while p_i in the cache do
    if p_i is in the confirmed area then // satisfy Equation (3)
        i ← i − 1;
    else if p_i is in the pruned area then // satisfy Equation (4)
        i ← i + 1; break;
    else if d(p_c, p_i) ≤ D then // p_i is in the check area
        i ← i − 1;
    else
        i ← i + 1; break;
if p_c.t − p_i.t ≥ T then
    return {p_i → p_{i+1} → ... → p_c} as a stay point;
else
    return ∅;
```

and a narrower check area. Interestingly, p_2 is located in the pruned area when $g = \frac{\sqrt{2}}{8}D$, so it does not need to be checked further.

Suppose $g = \frac{\sqrt{2}}{4n}D$, $n \geq 1$, we can express the confirmed area with:

$$(\Delta x + 1)^2 + (\Delta y + 1)^2 \leq 8n^2 \tag{3}$$

Similarly, the pruned area can be expressed by:

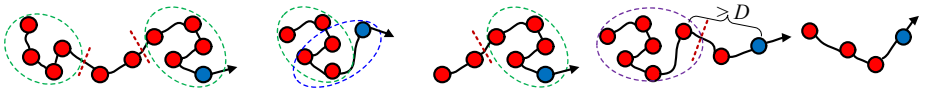
$$\begin{cases} (\Delta x - 1)^2 + (\Delta y - 1)^2 \geq 8n^2 & \text{if } \Delta x \geq 1 \text{ and } \Delta y \geq 1 \\ (\Delta y - 1)^2 \geq 8n^2 & \text{if } \Delta x = 0 \text{ and } \Delta y \geq 1 \\ (\Delta x - 1)^2 \geq 8n^2 & \text{if } \Delta x \geq 1 \text{ and } \Delta y = 0 \end{cases} \tag{4}$$

5 Stay Point Detection

The native stay point detection method is shown in Definition 4. There are two operations in stay point detection: identifying and merging.

Identifying. With the grid index, we can modify the naive algorithm to Algorithm 1. For each p_i in the cache, if p_i is in the confirmed area, we continue to check the previous point. If p_i is in the pruned area, we stop the check process directly and break out of the loop. If p_i is in the check area, we perform the same logic as the naive algorithm.

Merging. After identifying a stay point, we may merge it with the previous stay point if they share some GPS points. To avoid the expensive set join operation,



(a) Case 1.1 Two Stay Points Separated (b) Case 1.2 Two Stay Points Intersected (c) Case 1.3 Only One Stay Point (d) Case 2.1 Far Away from the Stay Point (e) Case 2.3 No Stay Point

Fig. 5: Trajectory Segmentation.

we simply mark the start and end positions of the latest stay point, denoted by I_s and I_e, respectively. Suppose the newly formed stay point has a start position I'_s and an end position I'_e. If $I'_s \leq I_e$, it means the newly formed stay point overlaps with the previous stay point, so we update I_e by setting $I_e = I'_e$ (thus it completes the merging process). Otherwise, we set $I_s = I'_s$ and $I_e = I'_e$ after performing a segmentation operation (see the next section).

6 Trajectory Segmentation

When a new GPS point is arriving, there are two cases. **Case 1**: the new GPS point forms a stay point with the previous GPS points; **Case 2**: the new GPS point does not belong to any stay point.

For Case 1, there are further three subcases. **Case 1.1**, there are two stay points including the newly formed one in the memory, but they are separated, as shown in Fig. 5(a). In this case, we segment the trajectory into at most three sub-trajectories, where the first stay point forms the first sub-trajectory, the "middle" GPS points (if any) that do not belong to any stay point form the second sub-trajectory, and the new stay point forms the third sub-trajectory. The first and second sub-trajectories are finally flushed out of the memory. **Case 1.2**, there are two stay points in the memory but they are intersected, as shown in Fig. 5(c). In this case, we simply merge the two stay points into one, and keep the merged stay point in memory. **Case 1.3**, there is only one stay point in the memory, as shown in Fig. 5(b). In this case, the trajectory is segmented into at most two sub-trajectories. The GPS points (if any) that do not belong to the stay point form the first sub-trajectory, which is then flushed. The new stay point forms the second sub-trajectory.

For Case 2, there are further three subcases. **Case 2.1**, as shown in Fig. 5(d), if there is already a stay point, but the newly arriving GPS point is far enough away from the last GPS point of the stay point, i.e., $d(p_c, p_{I_e}) > D$, we segment the trajectory into two sub-trajectories, where the stay point forms the first sub-trajectory, and the left points form the second sub-trajectory. The first trajectory is then flushed. In this case, we can still leverage the grid index to accelerate the calculation of the distance between p_c and p_{I_e}. **Case 2.2**, there is already a stay point, but $d(p_c, p_{I_e}) \leq D$. In this case, we do nothing. **Case 2.3**, there is no stay point yet in the memory, as shown in Fig. 5(e). In this case, we do nothing since the recent sub-trajectory is open.

7 Experiments

7.1 Datasets and Experimental Settings

Datasets. We adopt three real trajectory datasets from various domains, i.e., Chongqing taxi (CQ-Taxi), Chongqing hazardous chemical vehicle (CQ-Hazs) and Geolife [18]. CQ-Taxi contains 12,012 taxis and 69,110,069 GPS points from March 1, 2017 to March 2, 2017, in the city of Chongqing, China. Its sampling rate is 15 seconds. CQ-Hazs describes 6,068 hazardous chemical trucks and 42,824,668 GPS points from February 23 to February 28, 2022, in the city of Chongqing, China. Geolife contains 182 users, and 9,999,585 GPS points from April 18, 2009 to October 6, 2018, with the sampling rate varying from 3 seconds to 10 seconds in the city of Beijing, China.

Parameters. We study the effects of two parameters, as shown in Table 1, where the default values are marked in bold. They are the maximum distance parameter D, and the minimum time interval parameter T, respectively. D and T directly affect the segmentation results.

Table 1: Parameters

Parameters	**Values**
Distance threshold D	10, 20, **50**, 75, 100
Time threshold T	1, 3, **5**, 10, 15

Comparing Methods. We compare STEP with one representative interpolation-based segmentation algorithm SWS [3] and the existing typical stay point-based segmentation algorithm SPD [15, 19]. All source codes are publicly released [2].

7.2 Overall Comparison with Baselines

Fig. 6 presents the performance of different methods on three datasets, from which we have the following observation: STEP has the best performance on the three datasets in terms both of latency and throughput. Specifically, compared with SPD on the three datasets, STEP has only 34.7%, 14.1% and 17.0% latency, but achieves about 3x, 7x and 5.8x throughput. This improvement owes to the proposed grid index. SWS has longer latency and smaller throughput than STEP for all three datasets than STEP. SWS must wait until it reaches the following two GPS points before starting the trajectory segmentation, resulting in increased latency. Besides, it incorporates two time-consuming regression models for GPS point prediction, which leads to lower throughput.

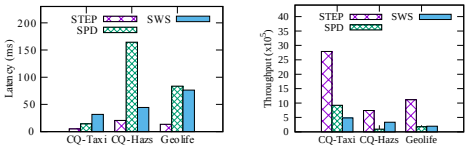
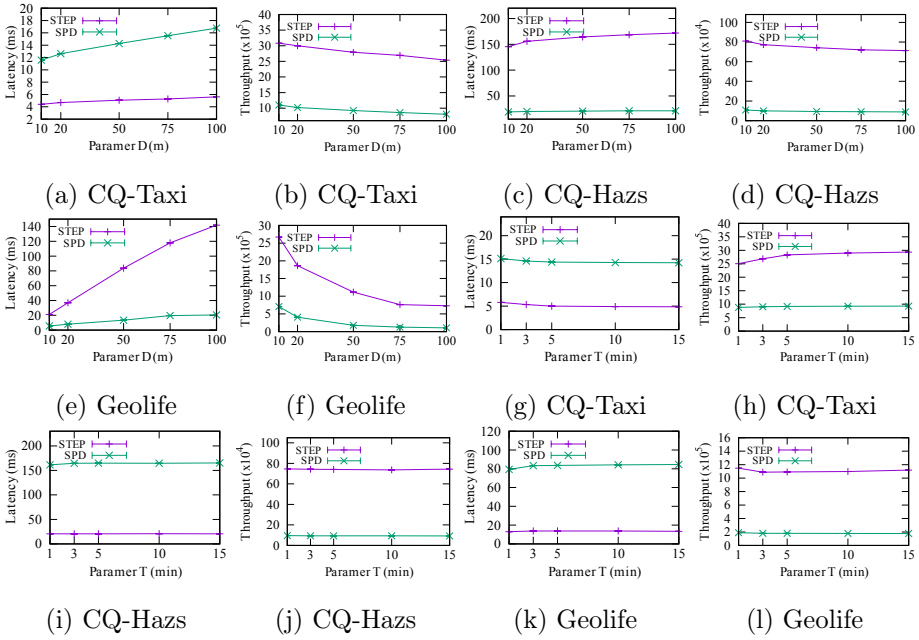


Fig. 6: Overall Comparison.

7.3 Performance with Different Parameters

When the distance parameter D is set as the independent variable, the experimental results are shown in Fig. 7(a)-(f). We can see from the figures that, with

[2] https://github.com/Spatio-Temporal-Lab/StreamingTrajSegment



(a) CQ-Taxi (b) CQ-Taxi (c) CQ-Hazs (d) CQ-Hazs

(e) Geolife (f) Geolife (g) CQ-Taxi (h) CQ-Taxi

(i) CQ-Hazs (j) CQ-Hazs (k) Geolife (l) Geolife

Fig. 7: Performance with Different Distance and Time Thresholds.

an increasing D, the performance of STEP and SPD decreases, because a larger D means more stay points can be identified, which brings about more operations of stay points merging and trajectory segmentation. However, the performance reduction of STEP is slower than that of SPD.

As shown in Fig. 7(g)-(l), we compare the performance of STEP and SPD with different values of T. With an increasing T, the performance of STEP and SPD is enhanced slightly, because a larger T indicates a stricter criteria for stay point formulation, leading to fewer stay points and less computation overhead.

8 Conclusion

This paper proposes an efficient streaming trajectory segmentation algorithm based on stay point detection, i.e., STEP. Extensive experiments using three different types of datasets ranging from taxis to trucks to pedestrians verify the powerful performance of STEP. In particular, STEP has only average 22% latency, but achieves about 5.3x throughput than SPD thanks to the carefully designed grid index. In our future work, we plan to explore more streaming applications after segmenting trajectory streams.

9 Acknowledgment

This paper is supported by the National Natural Science Foundation of China (62202070, 62322601), China Postdoctoral Science Foundation (2022M720567) and the Excellent Youth Foundation of Chongqing (CSTB2023NSCQJQX0025).

References

1. Cui, Z., Henrickson, K., Ke, R., Wang, Y.: Traffic graph convolutional recurrent neural network: A deep learning framework for network-scale traffic learning and forecasting. TITS **21**(11), 4883–4894 (2019)
2. Etemad, M., Júnior, A.S., Hoseyni, A., Rose, J., Matwin, S.: A trajectory segmentation algorithm based on interpolation-based change detection strategies. In: EDBT/ICDT Workshops. p. 58 (2019)
3. Etemad, M., Soares, A., Etemad, E., Rose, J., Torgo, L., Matwin, S.: Sws: an unsupervised trajectory segmentation algorithm based on change detection with interpolation kernels. GeoInformatica **25**, 269–289 (2021)
4. He, H., Li, R., Ruan, S., He, T., Bao, J., Li, T., Zheng, Y.: Trass: Efficient trajectory similarity search based on key-value data stores. In: ICDE. pp. 2306–2318. IEEE (2022)
5. He, H., Xu, Z., Li, R., Bao, J., Li, T., Zheng, Y.: Tman: A high-performance trajectory data management system based on key-value stores. In: ICDE. IEEE (2024)
6. Li, R., Bao, J., He, H., Ruan, S., He, T., Hong, L., Jiang, Z., Zheng, Y.: Discovering real-time reachable area using trajectory connections. In: DASFAA. pp. 36–53. Springer (2020)
7. Li, R., He, H., Wang, R., Ruan, S., He, T., Bao, J., Zhang, J., Hong, L., Zheng, Y.: Trajmesa: A distributed nosql-based trajectory data management system. IEEE Transactions on Knowledge and Data Engineering **35**(1), 1013–1027 (2021)
8. Li, R., He, H., Wang, R., Ruan, S., Sui, Y., Bao, J., Zheng, Y.: Trajmesa: A distributed nosql storage engine for big trajectory data. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE). pp. 2002–2005. IEEE (2020)
9. Li, R., Ruan, S., Bao, J., Li, Y., Wu, Y., Hong, L., Zheng, Y.: Efficient path query processing over massive trajectories on the cloud. TBD **6**(1), 66–79 (2018)
10. Li, R., Ruan, S., Bao, J., Li, Y., Wu, Y., Zheng, Y.: Querying massive trajectories by path on the cloud. In: ACM SIGSPATIAL. pp. 1–4 (2017)
11. Li, R., Zhu, H., Wang, R., Chen, C., Zheng, Y.: Fast and distributed map-matching based on contraction hierarchies. Journal of Computer Research and Development **59**(2), 342–361 (2022)
12. Markos, C., James, J., Da Xu, R.Y.: Capturing uncertainty in unsupervised gps trajectory segmentation using bayesian deep learning. In: AAAI. vol. 35, pp. 390–398 (2021)
13. Resheff, Y.S.: Online trajectory segmentation and summary with applications to visualization and retrieval. In: Big Data. pp. 1832–1840. IEEE (2016)
14. Ruan, S., Li, R., Bao, J., He, T., Zheng, Y.: Cloudtp: A cloud-based flexible trajectory preprocessing framework. In: ICDE. pp. 1601–1604. IEEE (2018)
15. Ruan, S., Long, C., Yang, X., He, T., Li, R., Bao, J., Chen, Y., Wu, S., Cui, J., Zheng, Y.: Discovering actual delivery locations from mis-annotated couriers' trajectories. In: ICDE. pp. 3241–3253. IEEE (2022)
16. Zaman, B., Altan, D., Marijan, D., Kholodna, T.: Reactive buffering window trajectory segmentation: Rbw-ts. Journal of Big Data **10**(1), 123 (2023)
17. Zheng, Y.: Trajectory data mining: an overview. TIST **6**(3), 1–41 (2015)
18. Zheng, Y., Xie, X., Ma, W.Y., et al.: Geolife: A collaborative social networking service among user, location and trajectory. IEEE Data Eng. Bull. **33**(2), 32–39 (2010)
19. Zheng, Y., Zhang, L., Ma, Z., Xie, X., Ma, W.Y.: Recommending friends and locations based on individual location history. TWEB **5**(1), 1–44 (2011)