The VLDB Journal (2024) 33:685–702
https://doi.org/10.1007/s00778-023-00833-w

REGULAR PAPER

Sub-trajectory clustering with deep reinforcement learning

Anqi Liang[1] · Bin Yao[1,3] · Bo Wang[1] · Yinpei Liu[2] · Zhida Chen[2] · Jiong Xie[2] · Feifei Li[2]

Received: 8 May 2023 / Revised: 4 December 2023 / Accepted: 10 December 2023 / Published online: 25 January 2024

Abstract
Sub-trajectory clustering is a fundamental problem in many trajectory applications. Existing approaches usually divide the clustering procedure into two phases: segmenting trajectories into sub-trajectories and then clustering these sub-trajectories. However, researchers need to develop complex human-crafted segmentation rules for specific applications, making the clustering results sensitive to the segmentation rules and lacking in generality. To solve this problem, we propose a novel algorithm using the clustering results to guide the segmentation, which is based on reinforcement learning (RL). The novelty is that the segmentation and clustering components cooperate closely and improve each other continuously to yield better clustering results. To devise our RL-based algorithm, we model the procedure of trajectory segmentation as a Markov decision process (MDP). We apply Deep-Q-Network (DQN) learning to train an RL model for the segmentation and achieve excellent clustering results. Experimental results on real datasets demonstrate the superior performance of the proposed RL-based approach over state-of-the-art methods.

Keywords Sub-trajectory clustering · Spatio-temporal similarity · Reinforcement learning · Deep learning

1 Introduction

With the proliferation of GPS-equipped devices and location-based services, a vast amount of trajectory data is generated at an unprecedented rate. Trajectory data capture the movements of moving objects and are highly valuable for various applications such as site selection, green transport, tourism planning, and traffic monitoring [19, 29, 35, 36]. In recent years, sub-trajectory clustering has attracted much attention. The sub-trajectory clustering task is to segment the trajectories into sub-trajectories based on certain principles and then cluster them w.r.t a given distance metric. Clustering the whole trajectories poses a challenge in discovering local similarities among trajectories since they vary in length and time range, leading to the loss of valuable information. As illustrated in Fig. 1, the four trajectories share similar sub-trajectories in the box, indicating the existence of a common pattern among them. However, clustering these trajectories as a whole will make the pattern unobservable because they are dissimilar and will be put into different clusters. Sub-trajectory clustering is helpful in many real-world applications, particularly for analyzing the correlations or common patterns among portions of different trajectories in regions of interest. For example, in hurricane forecasting, given the historical hurricane trajectories, researchers focus on the common patterns of sub-trajectories near the coastline because they can predict the locations of hurricane landfall by analyzing the patterns. Therefore, discovering the common patterns of sub-trajectories can improve the accuracy of hurricane forecasting and reduce damages caused by the hurricane.

✉ Bin Yao
yaobin@cs.sjtu.edu.cn

Anqi Liang
lianganqi@sjtu.edu.cn

Bo Wang
Wangbo324@sjtu.edu.cn

Yinpei Liu
yinpei.lyp@alibaba-inc.com

Zhida Chen
zhida.chen@alibaba-inc.com

Jiong Xie
xiejiong.xj@alibaba-inc.com

Feifei Li
lifeifei@alibaba-inc.com

1 Shanghai Jiao Tong University, Shanghai, China

2 Alibaba Group, Hangzhou, China

3 Hangzhou Institute of Advanced Technology, Hangzhou, China

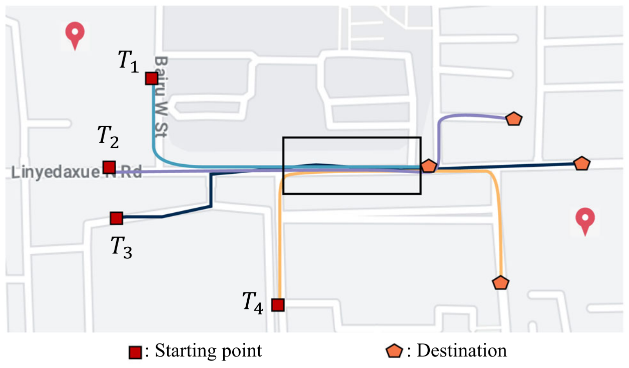


Fig. 1 Four dissimilar trajectories with a similar portion (the part in the black box)

Existing solutions to the sub-trajectory clustering problem follow the first-segment-then-cluster paradigm. The effectiveness of the clustering result heavily depends on the segmentation step, which is a crucial difference from the trajectory clustering problem. Existing solutions can be classified into two categories. The first one regards segmentation and clustering as two independent procedures. It segments the trajectories into sub-trajectories according to spatial features and then clusters these sub-trajectories. For instance, Lee et al. [18] proposed TRACLUS, which segments trajectories into line segments based on the minimum description length principle and then clusters line segments by applying DBSCAN. The second one segments trajectories by taking the clustering quality into account. For example, Pelekis et al. [25] segment trajectories based on the local density criterion and then conduct clustering according to the density. These methods usually rely on complicated hand-crafted segmentation rules tailored to specific applications, making the clustering result sensitive to these rules and limiting their generality.

To address these issues, we consider utilizing the clustering quality to guide the segmentation in a data-driven manner instead of using hand-crafted segmentation rules. A straightforward approach is enumerating all segmentation schemes and choosing the one with the best clustering quality. However, this method is computationally expensive and is thus impractical for real-world applications. Therefore, we propose an efficient and effective reinforcement learning (RL)-based framework that leverages clustering quality to guide segmentation. It allows segmentation and clustering components to cooperate closely, mutually enhancing each other and achieving superior clustering results.

Specifically, we treat the segmentation procedure as a sequential decision process, i.e., it sequentially scans the trajectory points and decides whether to segment at that point, which can be modeled as a Markov decision process (MDP) [26]. The crucial challenge of the MDP formulation is to define the state and reward, which significantly affect the performance of the learned model, i.e., the RL agent. Regarding the design of the state, the intuitive idea is to include more features to reflect the environment. However, experimental results indicate that such an approach may lead to much longer training time without noticeable improvements in the model's performance. To solve this problem, we propose a representative set of features that can effectively summarize the sub-trajectory clustering characteristics while maintaining acceptable training time. Designing an appropriate reward is vital to the training process as it encourages the RL agent to choose good actions to boost performance. We define the reward based on the evaluation metrics of clustering quality, which allows us to leverage the clustering quality to guide the trajectory segmentation, thus achieving the data-driven trajectory segmentation. We employ the Deep-Q-Network (DQN) method [24] to learn the policy for the MDP in our trajectory segmentation problem. Based on the learned policy, we propose a RL-based Sub-Trajectory Clustering algorithm called RLSTC. Our algorithm is data-driven and can adapt to different dynamics of the underlying trajectories. Moreover, it is a general solution that can be applied to various trajectory similarity measurements and application scenarios.

To further improve efficiency, we propose several optimizations to decrease the cost of trajectory distance computation. First, we implement a preprocessing step that simplifies trajectories by reserving only the critical points. Second, we apply an approximate method to compute trajectory similarity in linear time. Finally, we utilize an incremental similarity computation strategy to avoid recomputing the distance from scratch.

In summary, our main contributions are as follows:

- We propose a novel RL-based framework for the sub-trajectory clustering problem, which is the first RL-based solution to the best of our knowledge.
- We creatively model the procedure of trajectory segmentation as an MDP, especially defining the state and reward of MDP by considering the distinct features of the sub-trajectory clustering. This method achieves data-driven trajectory segmentation without relying on complex hand-crafted rules.
- We have developed several optimizations to enhance the efficiency of our solution.
- We conduct extensive experiments on various real-world trajectory datasets. The experimental results demonstrate that our RL-based algorithm outperforms state-of-the-art methods in terms of both effectiveness and efficiency.

The remainder of the paper is as follows: In Sect. 2, we present preliminaries and our problem statement. Section 3 discusses the preprocessing algorithm. In Sect. 4, we detail

Table 1 Symbols and description

Notation	Description
D	A dataset of N trajectories, i.e., $D = \{T_1, T_2, \cdots, T_N\}$
N	The number of trajectories in dataset D
T	A trajectory
p_i	The i^{th} point of T
$p(x)$	The longitude of point p
$p(y)$	The latitude of point p
$p(t)$	The timestamp of point p
$\|T\|$	The length of T, i.e., the number of points in T
$T(i, j)$	The sub-trajectory of T that starts and ends at the i^{th} and j^{th} point of T, respectively
$T(t_s)$	Starting time of T
$T(t_e)$	Ending time of T
s_t	State at time step t
a_t	Action taken at time step t
r_t	Reward obtained at time step t
θ	Parameters of the main neural network
θ'	Parameters of the target neural network
$\{C_j\}_{j=1}^k$	Learned k clusters
c_i	The center of cluster C_i
OD	Metric for evaluating the clustering quality

our RL-based algorithm, including modeling trajectory segmentation as an MDP and the algorithm itself. Experimental results are provided in Sect. 5. We give a brief review of related work in Sect. 6. Finally, Sect. 7 concludes the paper.

2 Preliminaries

In this section, we explain the preliminaries and present the definition of the sub-trajectory clustering problem. Commonly used notations are described in Table 1.

Definition 1 Trajectory A trajectory T is a sequence of time-ordered spatial points (points, for short), i.e., $T = \langle p_1, p_2, \cdots, p_n \rangle$, where p_i is a triplet (x_i, y_i, t_i). Here, (x_i, y_i) represents a location and t_i the timestamp. The variable n denotes the number of points in T. The length of T is defined by the number of points, i.e., $|T| = n$. The time interval of T is denoted by $[t_s, t_e]$, where t_s and t_e are the earliest and latest timestamp in T, respectively.

Definition 2 Sub-trajectory Given a trajectory $T = \langle p_1, p_2, \cdots, p_n \rangle$, a sub-trajectory $T(i, j) (1 \le i \le j \le n)$ of T is defined as $\langle p_i, \cdots, p_j \rangle$, representing that $T(i, j)$ starts at the i^{th} point and ends at the j^{th} point of T.

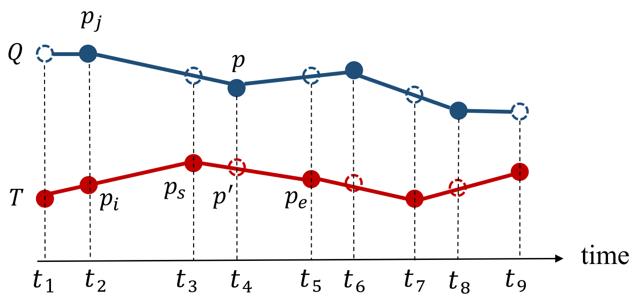


Fig. 2 An illustration of the synchronous point

Definition 3 Sub-trajectory Clustering A sub-trajectory cluster consists of sub-trajectories that are similar w.r.t a certain distance metric. The clustering procedure is to partition the sub-trajectories into k clusters, where k is a predefined parameter. Each cluster C_i has a center c_i. Note that c_i is not necessarily a sub-trajectory in C_i.

Definition 4 Clustering Quality Our objective is to cluster the sub-trajectories such that sub-trajectories in the same cluster are similar to each other. To measure the clustering quality, we adopt the popular agglomeration degree [27] and compute the overall distance of clusters as follows:

$$\mathrm{OD} = \sum_{i=1}^{k} \frac{m_i}{m} \mathrm{OD}_i, \tag{1}$$

where OD_i is the average distance between the center and the sub-trajectories in the cluster C_i, m_i is the number of sub-trajectories in C_i, and m is the total number of sub-trajectories. The computation of OD_i is given by Eq. (2):

$$\mathrm{OD}_i = \frac{\sum_{x \in C_i} d(x, c_i)}{m_i}. \tag{2}$$

Here, x is a sub-trajectory in cluster C_i, and $d(\cdot, \cdot)$ is a distance function that measures the distance between (sub-)trajectories.

By substituting OD_i in Eq. (1) with Eq. (2), OD becomes the average distance between each sub-trajectory and the cluster to which it belongs, thus allowing us to measure the effectiveness of the clustering. Our goal is to achieve clustering with a minimal OD. Note that any trajectory distance metric can be utilized in Eq. (2).

Definition 5 Synchronous point Consider any two trajectories T and Q, with p_i representing a point in T and p_j representing a point in Q. We define p_j as the synchronous point of p_i, and vice versa, if they share the same timestamp. As shown in Fig. 2, p_i and p_j form a pair of synchronous points.

For points on trajectory Q that lack a corresponding synchronous point on trajectory T, we establish synchronous points using linear interpolation. This process is detailed in Eqs. (3), (4), and (5). The dotted-line points illustrated in Fig. 2 represent the synchronous points obtained through this interpolation.

$$p'(x) = p_s(x) + \frac{p(t) - p_s(t)}{p_e(t) - p_s(t)} \cdot (p_e(x) - p_s(x)), \quad (3)$$

$$p'(y) = p_s(y) + \frac{p(t) - p_s(t)}{p_e(t) - p_s(t)} \cdot (p_e(y) - p_s(y)), \quad (4)$$

$$p'(t) = p(t). \quad (5)$$

We assume that any object corresponding to trajectory T does not move before $T(t_s)$ and after $T(e_s)$. For a point p in trajectory Q with a timestamp before $T(t_s)$, the location of p's corresponding synchronous point in trajectory T is set to the same as T's first point, and the timestamp of the synchronous point matches that of p. Similarly, if the timestamp of p is after $T(e_s)$, then the location of the synchronous point aligns with the last point in T, and its timestamp is set to be the same as that of p.

Definition 6 Trajectory Spatio-temporal Distance Spatial similarity is insufficient to evaluate the relationship between trajectories in real-world applications. Thus, we also consider temporal similarity. We define the spatio-temporal distance between two trajectories T and Q over a specific time interval $[t_{\min}, t_{\max}]$ by integrating their Euclidean distance over time:

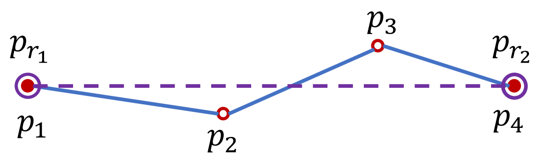
$$d_{\mathrm{IED}}(Q, T) = \int_{t_{\min}}^{t_{\max}} d_{Q,T}(t)\mathrm{d}t. \quad (6)$$

Here, $t_{\min} = \min(Q(t_s), T(t_s))$, $t_{\max} = \max(Q(t_e), T(t_e))$, with $d_{Q,T}(t)$ representing the Euclidean distance between synchronous points of the trajectories at time t, known as the synchronous Euclidean distance (SED) [23]. We perform linear interpolation on each point in T to find its synchronous point in Q and vice versa. Figure 2 illustrates this interpolation process, where solid circles represent original points and hollow ones are interpolated points. Therefore, Eq. (6) can be transformed into:

$$d_{\mathrm{IED}}(Q, T) = \sum_{k=1}^{k=n+m-1} \int_{t_k}^{t_{k+1}} d_{Q,T}(t)\mathrm{d}t, \quad (7)$$

where n and m are the number of points in Q and T, respectively, and t_k is the timestamp of an (interpolated) point. If the time intervals of T and Q do not intersect, $d_{\mathrm{IED}}(Q, T)$ is considered infinite.

We adopt the Trapezoid approximation method [13] to calculate the integral of $d_{Q,T}(t)$, which reduces the computation



$$L(H) = \log_2 d_{\mathrm{st}}(p_{r_1}, p_{r_2})$$
$$L(D|H) = \log_2(d_{\mathrm{IED}}(T, T_{\mathrm{simp}}))$$
$$T =< p_1, p_2, p_3, p_4 > \quad T_{\mathrm{simp}} =< p_{r_1}, p_{r_2} >$$

Fig. 3 Example of the $L(H)$ and $L(D|H)$ computation

cost to $O(n + m)$. The equation is as follows:

$$d_{\mathrm{IED}}(Q, T) = \frac{1}{2} \sum_{k=1}^{k=n+m-1} ((d_{Q,T}(t_k) + d_{Q,T}(t_{k+1})) \cdot (t_{k+1} - t_k)). \quad (8)$$

Furthermore, we employ an incremental calculation approach to reduce computational costs further. The distance between sub-trajectory $T(i, j)$ and trajectory Q can be computed as follows:

$$\begin{aligned} d_{\mathrm{IED}}(Q, T(i, j)) &= \frac{1}{2} \sum_{k=1}^{k=j-2} ((d_{Q,T}(t_k) + d_{Q,T}(t_{k+1})) \cdot (t_{k+1} - t_k)) \\ &+ \frac{1}{2}(d_{Q,T}(t_{j-1}) + d_{Q,T}(t_j)) \cdot (t_j - t_{j-1}). \end{aligned} \quad (9)$$

Here, the first term corresponds to the distance between $T(i, j - 1)$ and Q, a value that has already been computed. Thus, we only need to compute the second term, which has a time complexity of $O(1)$. Consequently, this approach reduces the time complexity from $O(m+n)$ to $O(1)$ through incremental calculation.

Problem 1 *Given a trajectory dataset D, the task of sub-trajectory clustering is to conduct segmentation on the trajectories in D and to form an optimal clustering* $C_{\mathrm{opt}} = \{C_j\}_{j=1}^{k}$ *such that Eq. (1) is minimized.*

3 Preprocessing

A preprocessing step is employed to simplify the trajectories, which is necessary for two main reasons. First, not all points in a trajectory are equally important. For example, when an object moves at a constant speed along a straight line, the points between the first and last ones do not carry significant information. Second, the cost of computing spatio-temporal similarity between trajectories is proportional to the number of points. Therefore, we retain only the significant points, referred to as *reserved points*, to simplify trajectories. The goal is to minimize the difference between the simplified and original trajectories while minimizing the number of points in the simplified trajectory. However, these two objectives are contradictory and require a trade-off. Inspired by [15, 18], we employ the minimum description length (MDL) principle to get an optimal simplification scheme. MDL consists of two components: $L(H)$ and $L(D|H)$. Here, $L(H)$ is the sum of the spatio-temporal lengths of all line segments in the simplified trajectory, and $L(D|H)$ represents the difference between the simplified and original trajectory. $L(H)$ denotes the degree of conciseness, and $L(D|H)$ the degree of preciseness. Suppose the original trajectory is $T = \langle p_1, p_2, \cdots, p_n \rangle$ and its simplified version is $T_{\text{simp}} = \langle p_{r_1}, p_{r_2}, \cdots, p_{r_m} \rangle$, where p_{r_i} denotes a reserved point in T. The MDL of T_{simp} and T is given by:

$$\begin{aligned}\text{MDL}_{T_{\text{simp}}} &= L(H) + L(D|H) \\ &= \log_2 \sum_{i=1}^{m-1} d_{\text{st}}(p_{r_i}, p_{r_{i+1}}) \\ &+ \log_2 \sum_{i=1}^{m-1} d_{\text{IED}}(T_i, T_{i_{\text{simp}}}),\end{aligned} \tag{10}$$

where $d_{\text{st}} = \frac{1}{2} d_{\text{ED}}(p_{r_i}, p_{r_{i+1}}) + \frac{1}{2} |p_{r_i}(t) - p_{r_{i+1}}(t)|$, and $d_{\text{ED}}(\cdot, \cdot)$ is the Euclidean distance between two points. $T_i = \langle p_i, \cdots, p_{r_{i+1}} \rangle$, and $T_{i_{\text{simp}}} = \langle p_{r_i}, p_{r_{i+1}} \rangle$.

$$\text{MDL}_T = L(H) + L(D|H) = \log_2 \sum_{i=1}^{n-1} d_{\text{st}}(p_i, p_{i+1}), \tag{11}$$

where the $L(D|H)$ of MDL_T is 0.

Example 1 As shown in Fig. 3, the original trajectory T is denoted by a solid line ($T = \langle p_1, p_2, p_3, p_4 \rangle$), and the simplified trajectory T_{simp} is represented by a dashed line ($T_{\text{simp}} = \langle p_{r_1}, p_{r_2} \rangle$), where p_{r_1} and p_{r_2} correspond to p_1 and p_4 in the original trajectory T, respectively. The MDL of T_{simp} and T is as follows:

$$\begin{aligned}\text{MDL}_{T_{\text{simp}}} &= \log_2 d_{\text{st}}(p_{r_1}, p_{r_2}) \\ &+ \log_2 d_{\text{IED}}(T, T_{\text{simp}}),\end{aligned} \tag{12}$$

$$\begin{aligned}\text{MDL}_T &= \log_2(d_{\text{st}}(p_1, p_2) + d_{\text{st}}(p_2, p_3) \\ &+ d_{\text{st}}(p_3, p_4)).\end{aligned} \tag{13}$$

Since each simplification scheme has its own MDL, we aim to determine the optimal scheme that yields the minimum MDL. However, obtaining an exact solution is impractical as it requires enumerating all combinations of reserved points

Algorithm 1: Trajectory preprocessing

```
Input  : A trajectory $T = \langle p_1, p_2, \cdots, p_n \rangle$
Output: A simplified trajectory $T_{\text{simp}}$
1  Initialize the set of reserved points as a empty list $RP \leftarrow [\,]$;
2  $i \leftarrow 0$, step $\leftarrow 1$;
3  Insert $p_1$ into $RP$;
4  while $i$ + step $< n$ do
5      $j \leftarrow i$ + step;
6      if $\text{MDL}_{T_{\text{simp}}(p_i, p_j)} > \text{MDL}_{T(p_i, p_j)}$ then
7          Insert $p_j$ into $RP$;
8          $i \leftarrow j$;
9          step $\leftarrow 1$;
10     else
11         step $\leftarrow$ step + 1;
12 Insert $p_n$ into $RP$;
13 Convert $RP$ into trajectory $T_{\text{simp}}$;
14 return $T_{\text{simp}}$;
```

in a trajectory. To address this, we propose an approximate greedy solution that regards the set of local optima as the global optimum. Assume the original trajectory T is represented as $\langle p_i, \cdots, p_j \rangle$. A local optimum is a simplified trajectory containing the minimum points while satisfying $\text{MDL}_{T_{\text{simp}}(p_i, p_k)} \leq \text{MDL}_{T(p_i, p_k)}$ for every k ($i < k \leq j$). Here, $T(p_i, p_k) = \langle p_i, \cdots, p_k \rangle$, and $T_{\text{simp}}(p_i, p_k) = \langle p_i, p_k \rangle$.

The preprocessing procedure is outlined in Algorithm 1. If $\text{MDL}_{T_{\text{simp}}(p_i, p_j)} > \text{MDL}_{T(p_i, p_j)}$, then p_j is chosen as a reserved point (Lines 6–9). Otherwise, p_j is omitted, and the next point is checked (Line 11). The time complexity of this algorithm is $O(n)$, where n is the number of points in T. This is because the number of MDL computations equals the number of points in T, and the time complexity of each MDL computation is $O(1)$ based on the incremental computation of d_{IED}.

4 Methodology

In this section, we first provide an overview of our RL-based solution in Sect. 4.1. We then detail how to model trajectory segmentation as an MDP in Sect. 4.2 and describe the learning procedure in Sect. 4.3. The computation of cluster centers is presented in Sect. 4.4. Finally, in Sect. 4.5, we introduce the RL-based Sub-Trajectory Clustering algorithm, termed as RLSTC.

4.1 Framework overview

Figure 4 presents the overview of our RL-based sub-trajectory clustering framework.

In our approach, we employ an offline training process, modeling the segmentation procedure as an MDP and

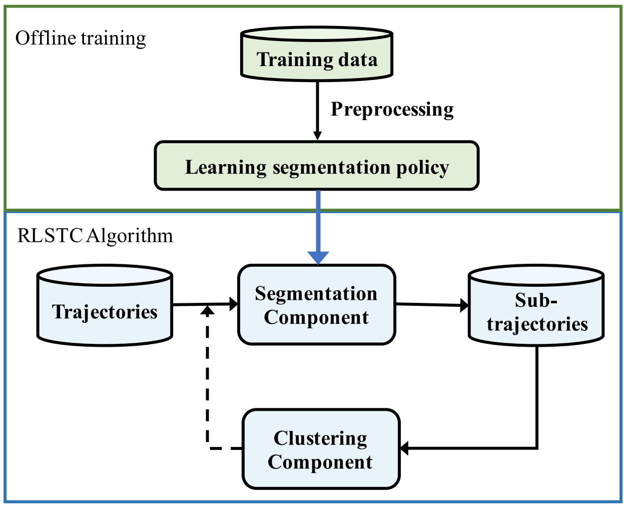


Fig. 4 Overview of the RL-based Solution

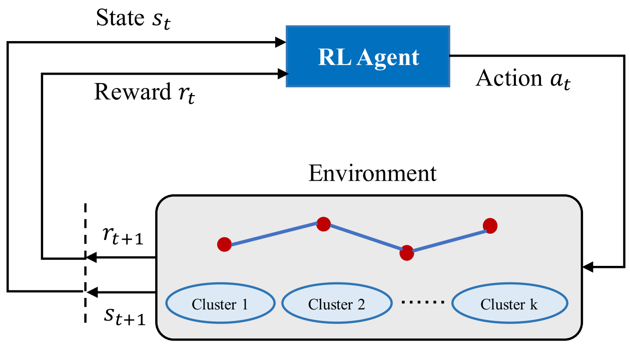


Fig. 5 Overview of the MDP formulation

applying the DQN method to learn an optimal segmentation policy. Subsequently, we develop the RLSTC algorithm, which comprises segmentation and clustering components. The segmentation component performs trajectory segmentation based on the learned policy, and then, the clustering component assigns each sub-trajectory to its nearest cluster and updates the cluster centers. After that, the clustering component forwards the updated cluster centers and clustering quality to the segmentation component, which leverages this information to generate improved segmentation. This iterative process continues until a satisfactory clustering result is achieved.

4.2 Modeling trajectory segmentation as MDP

Trajectory segmentation is a sequential decision-making task that involves scanning trajectory points to make segmentation decisions. Therefore, it can be effectively modeled as an MDP, as depicted in Fig. 5. In this process, the RL agent interacts with the environment by taking a sequence of actions. Each action changes the state of the RL agent, which receives a corresponding reward. We formulate the trajectory segmentation problem as an MDP with the following components:

States A state reflects the environment where decision-making is conducted. For instance, in a scenario with k cluster centers and a trajectory T, we denote the state at the t^{th} point of T as s_t. The challenge is determining the most appropriate features to represent this state accurately. The intuition is to include comprehensive features related to the trajectory clustering scenario, such as the cluster centers and trajectories. However, our experimental results show that this approach significantly slows down the training process and fails to yield satisfactory clustering results. An alternative design approach is to leverage the clustering quality to represent a state. This method involves calculating the OD values by assigning a new sub-trajectory to each cluster center and utilizing the resulting k OD values to represent the state. However, this design is constrained by the fact that a model trained for a specific k is not adaptable to different k values, thereby limiting its overall flexibility. Furthermore, our experiments reveal that this approach to defining the state leads to poor performance.

Therefore, we establish a comprehensive representation of the state by identifying a set of features that consider both the trajectories and the quality of clustering. In most clustering situations, the proximity of objects to the center of their respective clusters is a crucial indicator of effective clustering. Thus, newly generated sub-trajectories are assigned to the nearest cluster to ensure optimal clustering results. In this case, the OD at state s_t can be computed as:

$$s_t(\mathrm{OD}) = \frac{\sum_{x \in C_1} d(x, c_1) + \cdots + \sum_{x \in C_k} d(x, c_k)}{\mathrm{num}_{s_t}} = \frac{\sum_{i=1}^{\mathrm{num}_{s_t}} \min_{j=1,\cdots,k} \left\{ d(x_i, c_j) \right\}}{\mathrm{num}_{s_t}}, \tag{14}$$

where num_{s_t} is the number of sub-trajectories produced until s_t. We define two variables, OD_s and OD_n, to represent the OD produced by segmenting or not segmenting at the point currently being scanned.

Additionally, we employ an incremental approach that utilizes previously obtained OD values to enhance the efficiency of the computation process. The OD_s value of s_t can be computed as:

$$s_t(\mathrm{OD}_s) = \frac{s_{t-1}(\mathrm{OD}) \cdot \mathrm{num}_{s_{t-1}}}{\mathrm{num}_{s_{t-1}} + 1} + \frac{\min_{j=1,\cdots,k} \left\{ d(x, c_j) \right\}}{\mathrm{num}_{s_{t-1}} + 1}, \tag{15}$$

where x is the sub-trajectory produced by segmenting at p_t. The OD$_n$ of s_t can be computed as:

$$s_t(\mathrm{OD}_n) = s_{t-1}(\mathrm{OD}). \tag{16}$$

However, relying solely on this information might make the RL agent shortsighted. In other words, the agent might choose to segment at point p_t simply because $s_t(\mathrm{OD}_s) < s_t(\mathrm{OD}_n)$, potentially overlooking better results that could be achieved by segmenting at subsequent points after p_t. To address this, we introduce two variables, L_b and L_f, denoting the length of the generated sub-trajectory and the length of the remaining sub-trajectory, respectively. They are defined as:

$$s_t(L_b) = \frac{m_1}{|T|}, \quad s_t(L_f) = \frac{m_2}{|T|}, \tag{17}$$

where m_1 is the number of points in the generated sub-trajectory by segmenting p_t, and m_2 is the number of points in the sub-trajectory that starts at p_t and ends at the last point of T.

The convergence of the RL model can be expedited by integrating the expert knowledge, represented as OD$_b$ generated by TRACLUS. The OD$_b$ value can be calculated as:

$$\mathrm{OD}_b = \frac{\sum_{i=1}^{\mathrm{num}'} \min_{j=1,\cdots,k} \left\{ d(x_i', c_j) \right\}}{\mathrm{num}'}, \tag{18}$$

where num$'$ denotes the total number of sub-trajectories produced by segmenting trajectories in dataset D, and x_i' is the i^{th} sub-trajectory. Finally, the state s_t is defined as:

$$s_t = (s_t(\mathrm{OD}_s), s_t(\mathrm{OD}_n), \mathrm{OD}_b, s_t(L_b), s_t(L_f)). \tag{19}$$

Actions An action represents a decision made by the RL agent. Let p_t denote the current point to be considered. The action $a_t \in \{0, 1\}$ indicates whether to perform a segmentation at p_t. If $a_t = 1$, the agent executes a segmentation operation at p_t, thereby creating a new sub-trajectory and increasing the total number of sub-trajectories by one. In addition, the value of OD is updated accordingly. Conversely, if $a_t = 0$, the agent does not perform segmentation at p_t and instead proceeds to consider the next point p_{t+1}.

Transitions A transition refers to a change in the state resulting from an action taken by the agent. In trajectory segmentation, when the agent takes an action a_t in the current state s_t, it leads to a new state s_{t+1}. Since the absence of a known probability for the transition from s_t to s_{t+1}, we employ a model-free method to solve the MDP, which is particularly effective in scenarios where the transition probabilities are unknown.

Rewards The reward function assigns a value to a transition, reflecting the quality of the action that caused the transition. In order to guide the trajectory segmentation process using clustering quality, we utilize the OD value to define our reward function. Assuming a state transition from s_t to s_{t+1} after taking an action a_t, we define an immediate reward as the difference in OD values between successive states, i.e., $r_t = s_t(\mathrm{OD}) - s_{t+1}(\mathrm{OD})$. For each trajectory, when processing a sequence of states $s_1, s_2, \cdots, s_{|T|}$, the accumulative reward is defined as:

$$\begin{aligned} R &= \sum_{t=1}^{|T|-1} r_t \\ &= \sum_{t=1}^{|T|-1} (s_t(\mathrm{OD}) - s_{t+1}(\mathrm{OD})) \\ &= s_1(\mathrm{OD}) - s_{|T|}(\mathrm{OD}). \end{aligned} \tag{20}$$

where $s_1(\mathrm{OD}) = 0$, and $s_{|T|}(\mathrm{OD})$ corresponds to the OD of the terminated state. The objective of the MDP solver is to maximize the accumulative reward, which is consistent with minimizing OD.

4.3 Learning the optimal policy

RL aims to determine an optimal policy that specifies the action a to be taken in each state s to maximize the accumulative reward. The Q-value [40] is utilized to represent the total expected reward. The optimal Q-value, denoted as $Q^*(s, a)$, signifies the maximum accumulative reward achieved by following the optimal policy. According to the Bellman Expectation Equation [31], we have:

$$Q^*(s, a) = \mathbb{E}\left\{ r_t + \gamma \max_{a'} Q^*(s_{t+1}, a') | s_t, a_t \right\}, \tag{21}$$

which means that the maximum expected accumulative reward is the sum of the agent's immediate and the maximum future rewards. Here, γ is a discount factor that balances the importance of immediate and future rewards. We adopt the DQN method to approximate the optimal $Q^*(s, a)$ using a deep neural network $Q^*(s, a; \theta)$, where θ represents the set of parameters for the neural network.

Figure 6 shows the procedure of our DQN algorithm, and Algorithm 2 presents the pseudocode. The algorithm begins by initializing the main network $Q(s, a; \theta)$, the target network $\hat{Q}(s, a; \theta')$, a fixed-size replay memory pool M, and k cluster centers derived by the k-means++ method (Lines 1–4). The main network estimates the Q-value, and the target network computes the loss function for training the main network. Replay memory M stores recent transitions, which are uniformly sampled during training to reduce the correlation between consecutive transitions. The subsequent steps

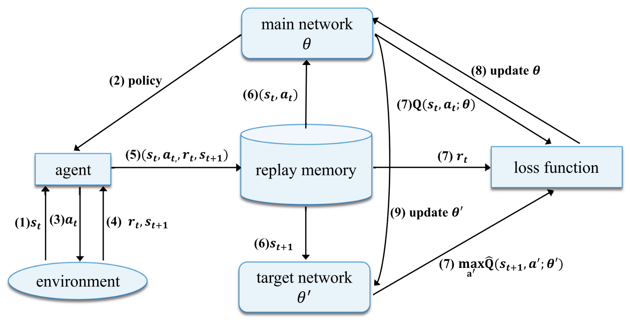


Fig. 6 Illustration of DQN learning

Algorithm 2: Deep-Q-Network (DQN) learning

```
Input  : A trajectory dataset $D$
Output: Learned Q-value function $Q(s, a; \theta)$
1  Initialize the main network $Q(s, a; \theta)$ with random parameters $\theta$;
2  Initialize the target network $\hat{Q}(s, a; \theta')$ with parameters $\theta' = \theta$;
3  Initialize the replay memory as a empty list $M \leftarrow [\,]$;
4  Initialize $k$ cluster centers by k-means++ method;
5  for epoch $= 1, 2, \cdots, m$ do
6      Shuffle the trajectory dataset;
7      for $i = 1, 2, \cdots, N$ do
8          Sequentially access trajectory $T_i$;
9          num $\leftarrow 0$;
10         /* $s_p$ is the latest segmentation point, and subtraj is the
           latest sub-trajectory */
11         $s_p \leftarrow p_1$, subtraj $\leftarrow \langle\,\rangle$;
12         $\mathrm{OD}_s \leftarrow 0$, $\mathrm{OD}_n \leftarrow 0$;
13         Compute $\mathrm{OD}_b$ by Eq. (18);
14         $L_b \leftarrow \frac{1}{|T_i|}$, $L_f \leftarrow 1$;
15         $s_1 \leftarrow (\mathrm{OD}_s, \mathrm{OD}_n, \mathrm{OD}_b, L_b, L_f)$;
16         for $t = 1, 2, \cdots, |T_i|$ do
17             Choose an action by $\varepsilon$-greedy strategy based on the
               main network;
18             if $t = |T_i|$ then
19                 subtraj $\leftarrow \langle s_p, \cdots, p_t \rangle$;
20                 break;
21             if $a_t = 1$ then
22                 subtraj $\leftarrow \langle s_p, \cdots, p_t \rangle$;
23                 $s_p \leftarrow p_t$;
24                 num $\leftarrow$ num $+ 1$;
25                 $s_t(\mathrm{OD}) \leftarrow s_t(\mathrm{OD}_s)$;
26             Update $\mathrm{OD}_s, \mathrm{OD}_n, L_b, L_f$ by Eq. (15)–Eq. (17);
27             Next state $s_{t+1} \leftarrow (\mathrm{OD}_s, \mathrm{OD}_n, \mathrm{OD}_b, L_b, L_f)$;
28             $r_t \leftarrow s_t(\mathrm{OD}) - s_{t+1}(\mathrm{OD})$;
29             $M$.append$((s_t, a_t, r_t, s_{t+1}))$;
30             Uniformly sample a minibatch from $M$;
31             Perform a stochastic gradient descent algorithm on the
               loss function;
32         Periodically update the target network $\hat{Q}(s, a; \theta')$;
33     Validate the model on validation dataset; Update cluster
       centers;
```

involve a series of episodes, each representing the segmentation process of a trajectory. Specifically, the algorithm shuffles trajectories at the start of the training epoch, processes each trajectory sequentially from the training dataset, and sets the initial state s_1 of the first trajectory point (Lines 6–15). The algorithm then proceeds with $|T_i|$ time steps. At each t^{th} time step, it checks point p_t and decides on an action using the ϵ-greedy strategy based on the main network (Line 17). If $a_t = 1$, segmentation is performed at p_t, and variables s_p, num, and $s_t(\mathrm{OD})$ are updated accordingly (Lines 22–25). If $a_t = 0$, the algorithm proceeds to point p_{t+1}. The algorithm observes a new state s_{t+1} and calculates the immediate reward r_t (Lines 26–28). Each experience tuple (s_t, a_t, r_t, s_{t+1}) is then added to replay memory M (Line 29). Periodically, the algorithm samples a random minibatch of experiences from M to update the main network using a stochastic gradient descent algorithm, aiming to minimize the mean squared error (MSE) (Lines 30–31), defined as:

$$\mathrm{MSE}(\theta) = (y - Q(s_t, a_t; \theta))^2, \tag{22}$$

where $Q(s_t, a_t; \theta)$ is the Q-value predicted by the main network, and y is the target Q-value, computed as:

$$y = \begin{cases} r_t, & \text{if } s_{t+1} \text{ is the terminated state} \\ r_t + \gamma \max\limits_{a'} \hat{Q}(s_{t+1}, a'; \theta'), & \text{otherwise.} \end{cases} \tag{23}$$

Here, $\hat{Q}(s_{t+1}, a'; \theta')$ represents the output of the target network. The target network $\hat{Q}(s, a; \theta')$ is periodically updated using the soft update method (Line 32). The optimal policy is learned by selecting the action a that maximizes $Q(s, a; \theta)$ for a given state s.

Moreover, we employ several strategies to minimize the impact of the traversal order of trajectories: (1) To mitigate the impact of the data accessing order, we shuffle the trajectories before each training epoch. (2) During training, we introduce randomness through an exploration-exploitation parameter, which allows the model to use its learned experiences while still trying unexplored actions, thus avoiding local optima. (3) We initialize the clustering centers using the k-means++ method to reflect the data distribution comprehensively. Additionally, we update the clustering centers after each training epoch to adapt to the evolving data.

4.4 Computation of cluster center

In this section, we propose a method for generating the representative trajectory, i.e., the cluster center. The cluster center captures the collective movement pattern of trajectories within the cluster. To achieve this, we determine the average coordinate at each timestamp to generate a representative trajectory. We scan the timestamps in chronological order and record the number of trajectories containing each timestamp. If the number of trajectories at a given timestamp is no less than a predefined threshold MinNum, we compute

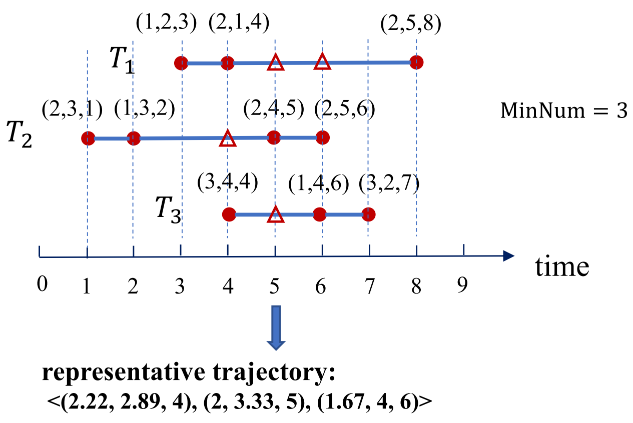


Fig. 7 Example of center computation

Algorithm 3: RLSTC algorithm

```
Input  : A trajectory dataset $D$, a threshold $\tau$, a learned model $Q(s, a; \theta)$, $k$
Output: $k$ clusters
1  Initialize $k$ cluster centers;
2  while true do
3      for episode $= 1, 2, \cdots, N$ do
4          Randomly sample a trajectory $T$ from $D$;
5          /* Segmentation component */
6          for $t = 1, 2, \cdots |T|$ do
7              $a_t \leftarrow \arg\max_a Q(s_t, a; \theta)$;
8              Execute Lines 18-27 of Algorithm 2;
9      /* Clustering component */
10     Assign each sub-trajectory to its nearest cluster;
11     Update the cluster centers of the $k$ clusters;
12     /* the new cluster centers are represented by $c_i$, the previous
       cluster centers in the last iteration are $c_i'$ */
13     centerdists $\leftarrow$ [ ];
14     for $i = 1, 2, \cdots, k$ do
15         centerdists.append($d(c_i, c_i')$);
16     maxdist $\leftarrow$ max(centerdists);
17     if maxdist $\geq \tau$ then
18         Forward the updated cluster centers and the current OD to
           the segmentation component;
19     if maxdist $< \tau$ then
20         return $k$ clusters;
```

the average coordinate for that timestamp. We derive synchronous points through linear interpolation for trajectories that lack a sampled point at that timestamp. Figure 7 provides a simple example of this approach, where the solid circles indicate the original sampled points, and the hollow triangles represent the synchronous points added through interpolation. The average coordinates at timestamps 4, 5, and 6 are computed because the number of trajectories with these timestamps meets or exceeds the threshold MinNum. This process results in the generation of the representative trajectory.

4.5 The RLSTC algorithm

In this section, we propose the RLSTC algorithm, outlined in Algorithm 3. Initially, we utilize the k-means++ method to initialize k cluster centers (Line 1). Subsequently, the learned policy is applied to segment each trajectory (Lines 6–8). The sub-trajectories are then assigned to their nearest cluster, and the cluster centers are updated accordingly (Lines 10–11). Following this, we calculate the maximum distance between the updated and previous cluster centers (Lines 14–16). If this maximum distance exceeds a predefined threshold, we forward the updated cluster centers and the clustering quality to the segmentation component (Lines 17–18). The algorithm terminates if the maximum distance does not exceed the threshold (Lines 19–20).

In Algorithm 3, the cluster centers will gradually converge after multiple iterations, regardless of the traversal order, thereby bringing a stable clustering result eventually. Therefore, the traversal order of trajectories has a limited impact on the clustering result.

Time Complexity The time complexity of the RLSTC algorithm is $O(nN)$, where n is the average length of trajectories, and N is the total number of trajectories in the dataset. Segmenting trajectories and assigning each sub-trajectory to its nearest cluster are the most time-consuming parts (Lines 3–10 in Algorithm 3). Assuming that the number of clusters, k, is a constant, the time cost for processing each trajectory point involves three components. First, computing the state requires $O(1)$ time, as the distance between a sub-trajectory and a cluster center is incrementally computed. Second, choosing an action, segmenting at a point, and assigning a sub-trajectory to its nearest cluster also take $O(1)$ time, as the minimum distance between the sub-trajectories and cluster centers is recorded while constructing the state. Finally, updating the state requires $O(1)$ time. Thus, the total time cost for processing all trajectories in the dataset is $O(nN)$. The time cost for updating the cluster centers (Line 11 in Algorithm 3), as described in Sect. 4.4, is $O(mM)$, where m is the average length of sub-trajectories, and M is the total number of sub-trajectories. Additionally, checking the difference between cluster centers of consecutive iterations has a time complexity of $O(m')$ (Line 16 in Algorithm 3), where m' is the average length of cluster centers. Therefore, the time complexity of one iteration is $O(nN + mM + m')$, and the total time complexity of RLSTC is $O(c(nN + mM + m'))$, with c representing the number of iterations. Since $O(nN)$ and $O(mM)$ are of the same order of magnitude, and $O(m')$ is less than $O(nN)$, and considering that RLSTC converges within a few iterations, the overall time complexity can be approximated as $O(nN)$.

5 Experiments

5.1 Experimental setup

Dataset We evaluate the performance of our algorithm on two widely used real-world trajectory datasets, Geolife and T-Drive.

1. Geolife [51] records the outdoor GPS trajectories of 182 users over three years. It contains 17,364 trajectories, covering a total distance of about 1.2 million kilometers and a duration of over 48,000 h. The GPS records were collected at a sampling rate of 1–5 s. To enhance computational efficiency, we constrain the number of points within each trajectory to between 10 and 500. Trajectories exceeding 500 points are randomly reduced to 500 points, and those with fewer than 10 points are discarded, resulting in a total of 17,070 trajectories.
2. T-Drive [47] contains the trajectories of 10,289 taxis in Beijing over a week. It comprises about 17 million points, with the total trajectory distance exceeding 9 million kilometers. The trajectory points were collected every 3–5 min. For trajectories with more than 500 points, we randomly select 500 points from them. Trajectories with fewer than 10 points are discarded. This process yields a total of 9,937 trajectories.

We divide the dataset into n parts, one for testing and the remaining $n-1$ parts for training. 10% of the training set is allocated as the validation set, while the remaining data are used for training. In our experiments, we set n to 5.
Evaluation platform We conduct all experiments on a Dell server with an NVIDIA P100 GPU, a 48-core Intel(R) Xeon(R) CPU E5-2678 v3 @ 2.50GHz, and 128GB memory. All methods are implemented using Python 3.6, and the implementation of our method is based on TensorFlow 2.2.0.

5.2 Baselines

We compare RLSTC with eight competitor methods in terms of clustering quality and efficiency.

- TRACLUS [18] is a widely adopted classical framework for trajectory segmenting and clustering, which utilizes artificial partitioning rules alongside density-based clustering to achieve effective results.
- S^{2}T-Clustering [25] employs a global voting algorithm that effectively adopts local density to segment and cluster trajectories.
- Sub-trajectory Clustering [1] (For convenience, we will refer to it as SubCLUS below.) SubCLUS is the state-of-the-art method that models the sub-trajectory clustering problem as a set-cover problem and proposes an approximation solution to achieve the optimization objective.
- Greedy Method is a degraded version of RLSTC that follows an optimal local strategy, segmenting at a point only if its OD$_s$ is smaller than its OD$_n$.
- CLUSTMOSA [8] uses three clustering objective criteria and applies archived multi-objective simulated annealing for optimal clustering.
- HyperCLUS [42] segments each trajectory into a series of hypercubes and identifies the common sub-trajectories (clusters) by evaluating intersections among hypercubes.
- RTCLUS [28] introduces a novel distance metric to improve segmentation and clustering accuracy, using an R-tree to enhance efficiency.
- RLSTC w/o simplification employs the same segmentation and clustering method as RLSTC but omits the trajectory preprocessing procedure.

5.3 Hyperparameter setting

The main network in DQN consists of a two-layer feedforward neural network. The first layer involves 64 neurons and uses the *ReLU* activation function. The second layer involves two neurons as the output, corresponding to the capacity of the action space. The target network has the same structure as the main network.

We utilize stochastic gradient descent (SGD) with a learning rate of 0.001 based on empirical findings. Hyperparameters were set following guidelines from RL-based trajectory processing literature [38, 39] and our preliminary experiments. The exploration-exploitation balance is controlled by the parameter ε, initially set to 1.0 and then reduced to 0.99ε after each step, with a lower bound of 0.1 to maintain a certain degree of exploration in the training. The reward discount factor γ is set as 0.99. The size of the replay memory and the sample size at each iteration are set as 5,000 and 32, respectively. The target network updates follow $\theta' = \omega\theta + (1-\omega)\theta'$ at the end of each episode, with ω set to 0.001. According to the preliminary experimental results, we set the cluster number k to 10 and the maximum distance threshold τ between cluster centers in successive iterations to 0.1.

To set the parameters of the baseline method TRACLUS, we use the heuristic method in [18]. If it fails to estimate the values of the parameters, we determine the parameters by manually searching over a range of values. The parameter settings for S^{2}T-Clustering, SubCLUS, CLUSTMOSA, HyperCLUS, and RTCLUS follow the strategies outlined in their respective studies [1, 8, 25, 28, 42].

5.4 Distance measurements

In addition to using d_{IED}, the default distance measurement presented in Definition 6, we evaluate RLSTC with three commonly used distance measurements. These are the discrete Fréchet distance d_{DF} [2], DTW distance d_{DTW} [45], and Weighted distance d_{WD} [18]. The equations for these distance measurements are as follows:

$$d_{\mathrm{DF}}(Q,T)=\begin{cases} d_{\mathrm{ED}}(p_1,q_1), \text{ if } |Q| \text{ and } |T| \text{ is } 1\\ \infty, \text{ if } Q=\varnothing \text{ or } T=\varnothing\\ \max(d_{\mathrm{ED}}(p_1,q_1),\\ \min(d_{\mathrm{DF}}(Q_h,T_h), d_{\mathrm{DF}}(Q,T_h),\\ d_{\mathrm{DF}}(Q_h,T))), \text{ otherwise,} \end{cases} \tag{24}$$

$$d_{\mathrm{DTW}}(Q,T)=\begin{cases} d_{\mathrm{ED}}(p_1,q_1), \text{ if } |Q| \text{ and } |T| \text{ is } 1\\ \infty, \text{ if } Q=\varnothing \text{ or } T=\varnothing\\ d_{\mathrm{ED}}(p_1,q_1)+\\ \min(d_{\mathrm{DTW}}(Q_h,T_h),\\ d_{\mathrm{DTW}}(Q,T_h),\\ d_{\mathrm{DTW}}(Q_h,T)), \text{ otherwise,} \end{cases} \tag{25}$$

where Q and T are trajectories, $d_{\mathrm{ED}}(\cdot,\cdot)$ computes Euclidean distance between two points, and Q_h (resp. T_h) represents the sub-trajectory starting from the second point of Q (resp. T).

$$\begin{aligned} d_{\mathrm{WD}}(Q,T) &= \omega_{\perp} d_{\perp}(Q,T)+\omega_{||} d_{||}(Q,T)\\ &\quad +\omega_{\theta} d_{\theta}(Q,T). \end{aligned} \tag{26}$$

Here, $d_{\perp}(\cdot,\cdot)$, $d_{||}(\cdot,\cdot)$, and $d_{\theta}(\cdot,\cdot)$ represent the perpendicular, parallel, and angular distances between trajectories, respectively, and $\omega_{\perp}$, $\omega_{||}$, and ω_{θ} are the corresponding weights set to 1.

5.5 Evaluation metrics

We utilize two widely recognized clustering metrics, namely OD (Eq. (1)) and SSE (Eq. (27)), to evaluate the effectiveness of sub-trajectory clustering. OD reflects the degree of agglomeration between objects and their corresponding cluster centers, which is highly relevant for center-based clustering methods such as agglomerative hierarchical clustering or k-means. On the other hand, SSE measures the proximity of objects within a cluster, making it more pertinent for density-based clustering methods like DBSCAN. By adopting these two metrics, we ensure a comprehensive and fair comparison between various clustering methods. In this paper, we calculate OD and SSE using trajectory distance measurements. Therefore, lower values of both metrics indicate higher clustering quality.

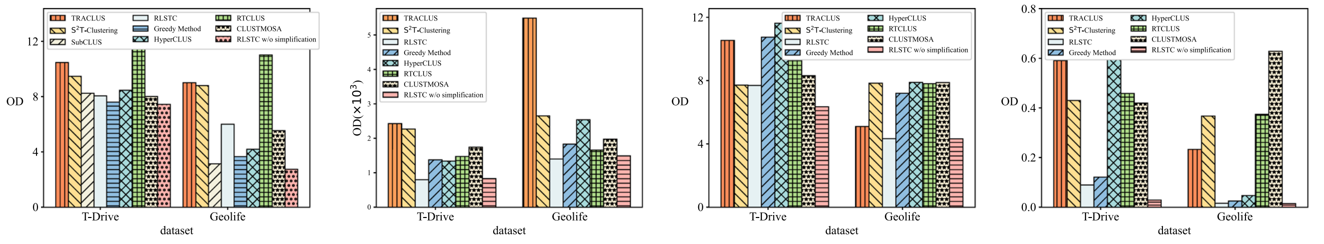
$$\mathrm{SSE}=\sum_{i=1}^{k}\left(\frac{1}{2|C_i|}\sum_{x\in C_i}\sum_{y\in C_i} d(x,y)^2\right). \tag{27}$$

5.6 Experimental results

Effectiveness We conducted experiments on T-Drive and Geolife using various sub-trajectory clustering methods and different distance measurements. To be noted, SubCLUS leverages properties more specific to the discrete Fréchet distance, enabling it to achieve an approximate solution while significantly accelerating the algorithm without compromising the clustering quality. Therefore, we exclusively evaluate SubCLUS using d_{DF} in our experiments. Since the large number of sub-trajectories generated by TRACLUS makes it less practical, e.g., 33,168 sub-trajectories for 1,000 trajectories, we evaluate these methods on 1,000 trajectories from the testing dataset. We apply the N-fold validation method and report the average clustering performance. Figures 8 and 9 show the OD and SSE of the clustering result for different methods, respectively. Our experiments demonstrate that RLSTC consistently outperforms baselines across different datasets and distance measurements in terms of all clustering quality metrics.

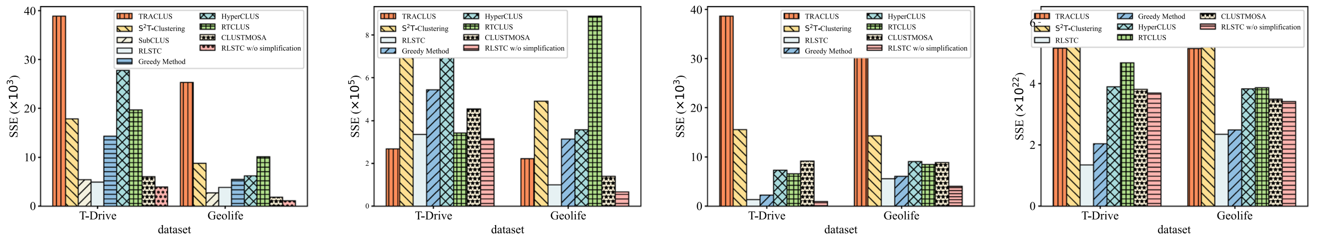
On average, RLSTC outperforms other methods by 36% on T-Drive and 39% on Geolife in terms of OD. For SSE, the improvements are 57% and 44% on T-Drive and Geolife, respectively. Notably, RLSTC outperforms TRACLUS, CLUSTMOSA, HyperCLUS, and RTCLUS, which separate segmentation and clustering phases and do not leverage clustering results to improve the trajectory segmentation. Furthermore, RLSTC outperforms S^{2}T-Clustering and SubCLUS, which rely on complex artificial rules. RLSTC also avoids the local optima caused by the shortsighted decision-making property of Greedy Method, demonstrating the effectiveness of reinforcement learning in sub-trajectory clustering. Additionally, the minor performance difference between RLSTC with and without simplification indicates that the trajectory simplification procedure has a minor impact on clustering quality.

Efficiency To evaluate the efficiency of different methods, we measured their running times on T-Drive and Geolife datasets, accounting for both segmentation and clustering time. In this experiment, we utilized the d_{IED} distance measurement. As shown in Fig. 10, RLSTC consistently runs faster than TRACLUS, S^{2}T-Clustering, SubCLUS, Greedy Method, and CLUSTMOSA, with at least 20% performance improvement. The efficiency improvement is even more apparent with larger trajectory sizes. For example, RLSTC outperforms TRACLUS and S^{2}T-Clustering by 37% and 24% on T-Drive, respectively, and it outperforms



(a) d_{DF} (b) d_{DTW} (c) d_{WD} (d) d_{IED}

Fig. 8 Clustering quality (OD) of different sub-trajectory clustering methods



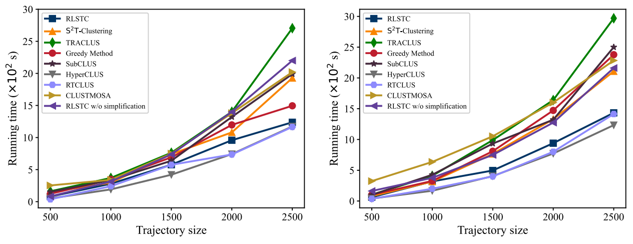
(a) d_{DF} (b) d_{DTW} (c) d_{WD} (d) d_{IED}

Fig. 9 Clustering quality (SSE) of different sub-trajectory clustering methods

SubCLUS and Greedy Method by 32% and 30% on Geolife, respectively. HyperCLUS and RTCLUS are slightly faster than RLSTC for the following reasons: HyperCLUS segments trajectories into hypercubes and performs clustering by checking intersections among hypercubes, which is faster than computing distances between raw trajectories. RTCLUS employs Minimum Bounding Rectangles (MBRs) to represent sub-trajectories and uses an R-tree index, accelerating the distance computation during clustering. However, both HyperCLUS and RTCLUS rely on approximate trajectory representations, neglecting precise calculations on trajectories. Consequently, they result in worse clustering performance, as shown in Figs. 8 and 9. It is worth noting that RLSTC w/o simplification runs much slower than RLSTC but remains faster than TRACLUS on Tdrive and faster than TRACLUS, S^{2}T-Clustering, CLUSTMOSA, and Greedy Method on Geolife, which demonstrates that the preprocessing step can significantly improve efficiency while maintaining satisfactory clustering quality. Considering the significant improvement in the clustering quality, RLSTC is a better choice than Greedy Method. Therefore, we do not include Greedy Method in the following experiments.

5.7 Effectiveness of segmentation methods

Our research primarily focuses on proposing a clustering-friendly trajectory segmentation method that can adjust the segmentation strategy based on clustering quality, applicable to both center- and density-based clustering methods. Therefore, our experiments aim to evaluate the performance of different segmentation methods regarding clustering quality.



(a) T-drive (b) Geolife

Fig. 10 Running time

To achieve this, we apply different segmentation approaches to generate sub-trajectories and then employ the same clustering algorithm on them. Specifically, we adopt the segmentation methods in RLSTC, TRACLUS, S^{2}T-Clustering, SubCLUS, HyperCLUS, RTCLUS, and CLUSTMOSA, denoted as RLSTC-S, TRACLUS-S, S^{2}T-Clustering-S, SubCLUS-S, HyperCLUS-S, RTCLUS-S, CLUSTMOSA-S, to produce sub-trajectories. We then measure their OD after running the same clustering algorithm. We run the above procedure for three different clustering algorithms: Agglomerative Hierarchical Clustering (AHC), DBSCAN, and k-means, covering approaches that specify a cluster number or are density-based. We use four distance measurements, i.e., d_{IED}, d_{DF}, d_{DTW}, and d_{WD}, during the segmentation and clustering phases. The experimental framework is illustrated in Fig. 11.

Notably, the large number of sub-trajectories generated by TRACLUS-S makes it less practical, e.g., 33,168 sub-trajectories for 1,000 trajectories. Therefore, we evaluated different segmentation methods on small datasets. The results

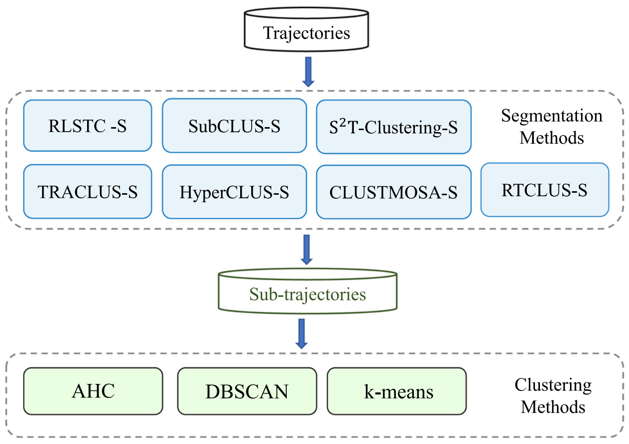


Fig. 11 Procedure of evaluating the segmentation methods

on Geolife are omitted because they yield similar observations.

As shown in Fig. 12, RLSTC-S consistently outperforms other segmentation methods across different clustering algorithms and distance measurements. In particular, RLSTC-S surpasses HyperCLUS-S, RTCLUS-S, CLUSTMOSA-S, TRACLUS-S, S^{2}T-Clustering-S, and SubCLUS-S by 46%, 47%, 45%, 44%, 33%, and 11%, respectively, implying that RLSTC-S outperforms the methods separating segmentation and clustering phases, e.g., TRACLUS-S, HyperCLUS-S, RTCLUS-S and CLUSTMOSA-S, and the methods depending on hand-crafted heuristics, e.g., S^{2}T-Clustering-S and SubCLUS-S. The results demonstrate the effectiveness of our RL-based segmentation approach, which can leverage clustering quality to guide trajectory segmentation.

Furthermore, the results indicate that RLSTC-S is adaptable to various clustering methods, whether they require specifying the number of clusters or are density-based. Therefore, RLSTC-S can be seamlessly integrated into different clustering algorithms, thereby enhancing their performance.

5.8 Parameter study

Impact of OD$_b$ In Sect. 4.2, we defined the concept of the state and introduced the expert knowledge OD$_b$ into it. This variable, OD$_\text{b}$, represents OD produced by TRACLUS. We conducted experiments using RLSTC on T-Drive and Geolife, comparing scenarios with and without OD$_b$ included in the state. The results, shown in Fig. 13, demonstrate that clustering performance is significantly better with OD$_b$ included in the state for both datasets. This improvement can be attributed to the fact that including OD$_b$ accelerates the convergence, leading to enhanced clustering performance.

Impact of training data size To evaluate the impact of training data size, we constructed five training datasets randomly sampled from T-Drive, consisting of 1,000, 2,000, 3,000, 4,000, and 5,000 trajectories, respectively. Additionally, we selected 1,000 trajectories from the remaining data for testing. We recorded each model's training time and performance, as shown in Figures 14a and b. The clustering quality slightly improves (i.e., the OD value decreases) when the training size increases, but so does the training cost. The OD value becomes stable as the training size further increases. Since our objective is achieving low OD within short training time, we define a metric TR in Eq. (28) to represent the trade-off between training time and OD:

$$\mathrm{TR} = t_{\mathrm{norm}} + \mathrm{OD}, \tag{28}$$

$$\text{where } t_{\mathrm{norm}} = t \cdot \frac{\overline{\mathrm{OD}}}{\overline{t}}. \tag{29}$$

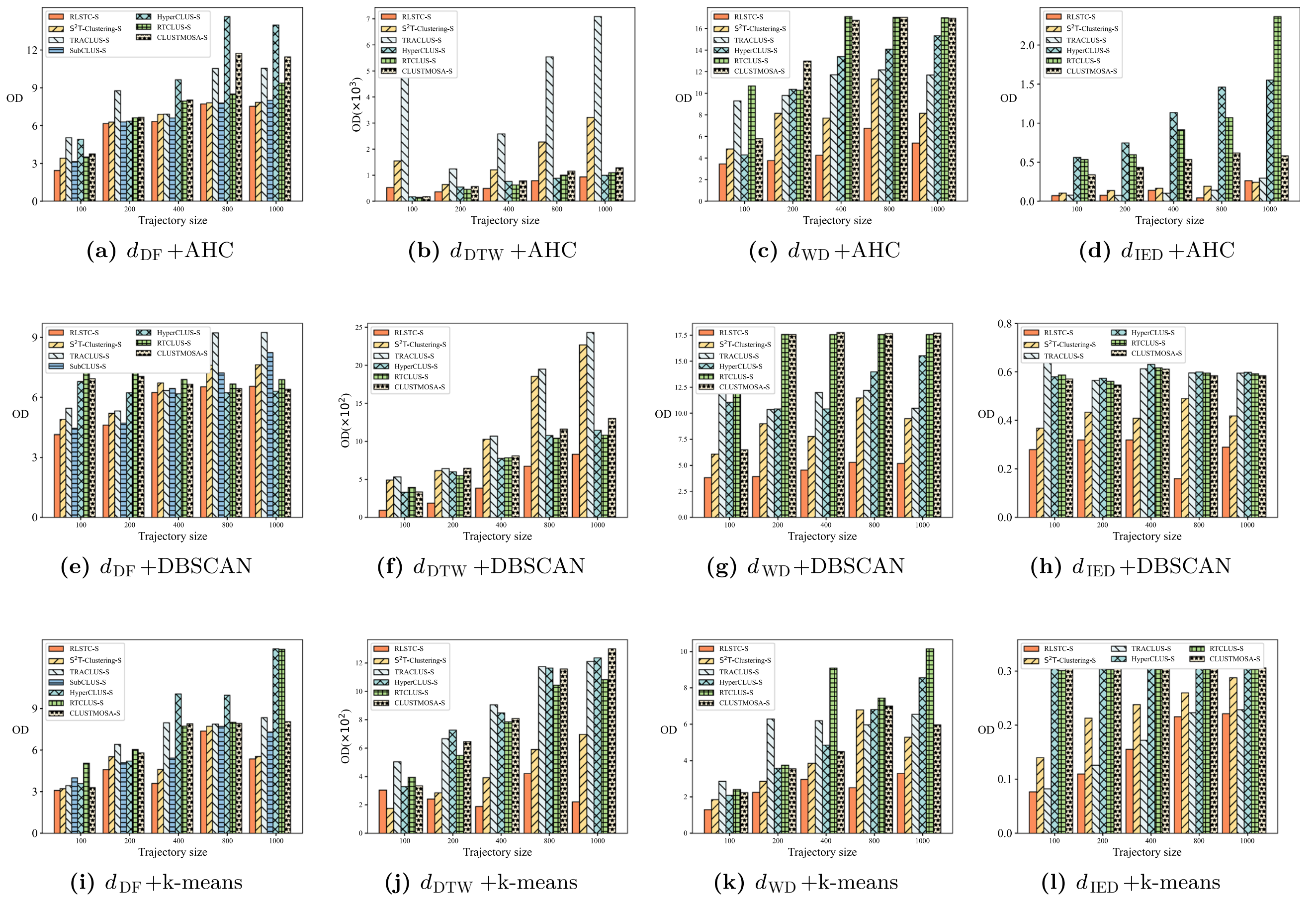
Eq. (29) normalizes the training time to the scale of OD, with t representing the training time. The optimal trade-off is achieved when TR is minimized. As shown in Figure 14c, TR is minimized when the training set size is 2,000. Consequently, we set the training set size as 2,000 in the remaining experiments.

Value of k We trained models using different k, i.e., $k \in \{5, 8, 10, 12, 14\}$, and evaluated them on the test dataset using the same k. Table 2 shows the results. For both datasets, the model trained using $k = 10$ achieves the best clustering quality.

Furthermore, we conducted an experiment to measure the clustering quality of a pretrained model against different k, which was trained using a certain k. Table 3 shows the results for a model trained using $k = 10$. The results reveal that when the difference between the tested k and the k for training is no more than 5, e.g., $k = 5, 8, 10, 12$, or 15, the variation in OD is small. In contrast, when the difference exceeds 5, e.g., $k = 20, 40, 60, 80$, or 100, there is a significant increase in the OD value. The results indicate that when the actual k value differs from the k for model training by more than 50%, the model's performance would deteriorate substantially. Based on our experimental findings, it is better to conduct a model retraining when the actual number of clusters differs from the k for model training by over 50%.

Additionally, as shown in Tables 3 and 4, we observe that a small sample can produce similar trends to the result on a complete dataset. Therefore, we can use a small dataset to check whether the model works well for the actual k. If the result reveals a substantial deviation in OD, e.g., over 40%, then we should conduct model retraining on the entire dataset. This approach offers a practical method for determining the usability of a model and when it is necessary to retrain the model.

Number of iterations We conducted experiments to explore the impact of the number of iterations by randomly sampling 1,000 trajectories from datasets. Figure 15a shows



(a) d_{DF} +AHC **(b)** d_{DTW} +AHC **(c)** d_{WD} +AHC **(d)** d_{IED} +AHC

(e) d_{DF} +DBSCAN **(f)** d_{DTW} +DBSCAN **(g)** d_{WD} +DBSCAN **(h)** d_{IED} +DBSCAN

(i) d_{DF} +k-means **(j)** d_{DTW} +k-means **(k)** d_{WD} +k-means **(l)** d_{IED} +k-means

Fig. 12 Performance of different segmentation methods

Table 2 Impact of *k*

k values	5	8	10	12	14
OD in T-Drive testing dataset	0.49	0.46	0.21	0.44	0.42
OD in Geolife testing dataset	0.052	0.051	0.023	0.051	0.04

Table 3 Clustering quality for a pretrained model ($k = 10$) against different *k* for testing

	$k = 5$	$k = 8$	$k = 10$	$k = 12$	$k = 15$	$k = 20$	$k = 40$	$k = 60$	$k = 80$	$k = 100$
OD in T-Drive dataset ($k = 10$)	0.25	0.22	0.21	0.26	0.26	0.51	0.59	0.61	0.60	0.62
OD in Geolife dataset ($k = 10$)	0.04	0.04	0.02	0.03	0.04	0.08	0.09	0.10	0.09	0.12

Table 4 Clustering quality for a pretrained model ($k = 10$) against different *k* on a small dataset containing 200 sampled trajectories

	$k = 5$	$k = 8$	$k = 10$	$k = 12$	$k = 15$	$k = 20$	$k = 40$	$k = 60$	$k = 80$	$k = 100$
OD in the small T-Drive dataset ($k = 10$)	0.14	0.19	0.18	0.18	0.19	0.32	0.34	0.34	0.35	0.37
OD in the small Geolife dataset ($k = 10$)	0.03	0.03	0.02	0.03	0.04	0.06	0.07	0.08	0.08	0.11

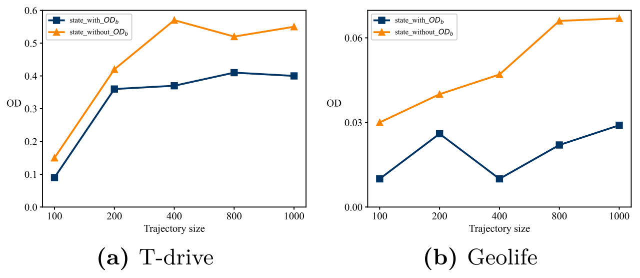


Fig. 13 Impact of OD_b

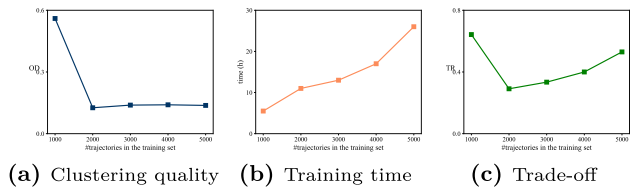


Fig. 14 Impact of training data size

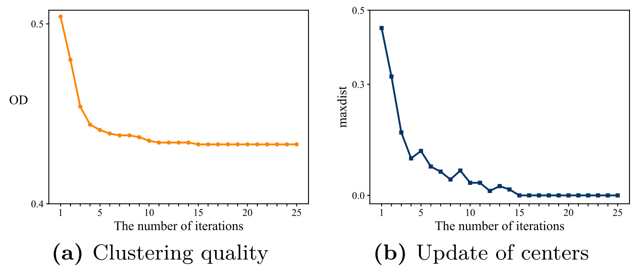


Fig. 15 Varying the number of iterations

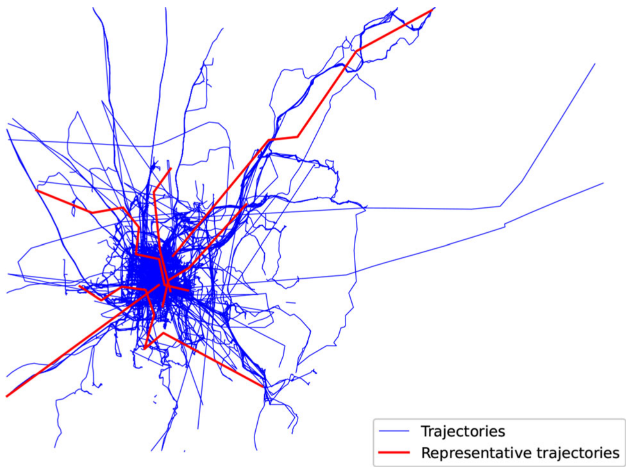


Fig. 16 Visualization on geolife

the results on T-Drive, while results on Geolife are omitted because they yield similar trends. Initially, the OD fluctuates during the first nine iterations but stabilizes after the 10^{th} iteration, indicating that the clustering performance gradually converges with increasing iterations. Furthermore, we examined the changes in cluster centers during the iterations. We used maxdist, denoted as τ in Algorithm 3, to represent the maximum distance between the cluster centers of consecutive iterations. The result is shown in Figure 15b. We can observe that the centers drastically change at the beginning, and then, they stabilize after the 10^{th} iteration, meaning that the clustering process has converged. The results demonstrate that RLSTC efficiently converges within a small number of iterations. Additionally, our experiments indicate that OD stabilizes when τ is set to 0.1.

5.9 Case study by visualization

The sub-trajectory clustering results on Geolife are visualized in Fig. 16, where thin blue lines depict trajectories, and thick red lines represent representative trajectories, i.e., cluster centers. The visualization shows that five clusters have been identified, with the cluster centers capturing common sub-trajectories of these clusters. Notably, the clustering centers exhibit a radial pattern, extending from the center to the outer areas, which is consistent with the distribution of the road network. Therefore, the result is reasonable and can be used to identify hot areas.

5.10 Discussion

The computation cost of trajectory distance is a bottleneck in our algorithm. Whenever a point p_i is scanned, it necessitates computing the distance between each cluster center and the sub-trajectory $\langle s_p, \cdots, p_i \rangle$ (where s_p denotes the previous segmentation point), resulting in high computation costs, particularly for lengthy trajectories. To address this challenge, we explore potential improvements to our algorithm. First, implementing a grid index can map trajectory points into grid cells, effectively reducing trajectory length and computational complexity. By approximating trajectory distance with the distance between sequences of grid cells, we can adjust grid cell size to achieve a balance between efficiency and accuracy. Second, to reduce the computational cost of evaluating each point as a candidate for segmentation, we propose augmenting the MDP with additional actions for skipping certain points. This approach leverages the fact that successive points often exhibit similar motion trends. For example, introducing a new action S into the action space would enable the algorithm to skip points $p_{i+1}, \cdots, p_{i+S}$ when scanning point p_i and proceed directly to scanning point p_{i+S+1}.

6 Related work

In this section, we briefly review the related work on trajectory clustering, sub-trajectory clustering, and deep reinforcement learning, respectively.

6.1 Trajectory clustering

Trajectory clustering groups trajectories into several clusters so that trajectories in the same cluster are more similar to each other than to those in different clusters. It plays a fundamental role in trajectory mining and analysis tasks [7, 17, 50] and has been extensively studied for decades. Most existing trajectory clustering algorithms define a similarity measurement and then employ classical clustering techniques, such as k-means [21], BIRCH [49], DBSCAN [9], and OPTICS [4], to group the trajectories. Gaffney et al. [14] proposed a clustering algorithm based on regression mixture models, which uses the Expectation-Maximization algorithm to cluster trajectories. Ferreira et al. [12] presented a trajectory clustering method called Vector Field k-means. They utilize vector fields to define and represent clusters and to indicate the similarity between trajectories. Afterward, they apply the classical k-means method for clustering. Wang et al. [34] investigated trajectory clustering on road networks and proposed a novel distance measurement and a scalable clustering method.

With the rapid development of deep learning, some researchers utilize deep learning methods to solve the trajectory clustering problem. For example, Yao et al. [44] transform each trajectory into a fixed-length feature vector via a sequence-to-sequence auto-encoder and then apply the k-means algorithm over the feature vectors. Wang et al. [37] employ network representation learning to learn the low-dimensional representation vectors of vehicle trajectories and then utilize machine learning methods to cluster the vehicle vectors. Fang et al. [11] introduced an end-to-end deep trajectory clustering framework, which is self-training without extracting manual features. The trajectory clustering problem does not consider trajectory segmentation, which is essential to the sub-trajectory clustering problem.

6.2 Sub-trajectory clustering

There are two main categories of work on the sub-trajectory clustering problem. The first line of work segments trajectories into sub-trajectories according to specific criteria such as location, direction, speed, and shape [3, 6, 10, 30], ensuring homogeneity within each sub-trajectory. Subsequently, a clustering algorithm is used to group these sub-trajectories. For instance, Lee et al. [18] proposed the TRACLUS framework, which includes partition and group phases. In the partition phase, trajectories are divided into sub-trajectories based on the MDL principle. In the group phase, density-based clustering method is performed over these sub-trajectories. Similarly, Li et al. [20] developed an incremental clustering algorithm building on TRACLUS. Dutta et al. [8] employed bearing measurements for trajectory segmentation to identify significant turning points based on the directional degree at each point. They then leverage an archived multi-objective simulated annealing approach for optimal clustering. Xia et al. [42] applied a trajectory segmentation approach considering location, time, and motion direction variations. They transform each trajectory into a sequence of hypercubes and identify common sub-trajectories by checking intersections among hypercubes for clustering. Qiao et al. [28] propose enhancements to TRACLUS, introducing motion direction as a segmentation criterion and devising a novel distance measurement while leveraging the R-tree index to improve efficiency in clustering sub-trajectories. In these methods, trajectory segmentation and clustering are independent, where clustering quality does not guide the segmentation. In addition, segmentation criteria are often designed for specific applications, resulting in a lack of generality. Buchin et al. [5] presented a sub-trajectories clustering algorithm to detect commuting patterns in trajectories. However, this method requires users to define additional cluster parameters, which can be challenging to set in advance.

Another research approach considers clustering quality during trajectory segmentation. Pelekis et al. [25] proposed the S^2T-Clustering method, which uses a global voting algorithm to indicate the local density of sub-trajectories. Trajectories are then segmented based on this local density, followed by clustering of the resulting sub-trajectories. However, this algorithm requires sophisticated preprocessing and a set of complex predefined parameters, leading to high computational costs. Tampakis et al. [33] expanded the S^2T-Clustering algorithm for distributed environments. Agarwal et al. [1] introduced *pathlet* to represent each sub-trajectory cluster, aiming to find the optimal collection of pathlets that best represent the trajectories. They present an objective function for this purpose but show that finding the optimal pathlet collection is an NP-hard problem. Hence, they proposed fast approximation algorithms. However, these algorithms rely on complex artificial rules and are limited to specific trajectory distance measurements. Zygouras et al. [52] explored identifying common paths frequently followed by the given trajectories. Tampakis et al. [32] and Zhang et al. [48] further researched the distributed sub-trajectory join and query algorithms, which aim at finding similar parts of trajectories with the query trajectory in real-time. Notably, the works of [1, 32, 48, 52] focus on identifying shared portions among trajectories, which differs from our research focus.

6.3 Reinforcement learning

Reinforcement learning (RL) is a method that enables an agent to learn from feedback through trial-and-error interactions with the environment. In RL, the problem to solve is described as an MDP. RL has been widely applied in various fields, such as database optimization, traffic light management, index construction, and trajectory analysis. In database optimization, Yu et al. [46] presented a novel learned optimizer utilizing RL with Tree-structured LSTM for join order selection. Marcus et al. [22] combined tree convolutional neural networks with an RL algorithm to design a query optimizer. In index construction, Yang et al. [43] developed a qd-tree index based on RL methods to reduce I/O costs. Gu et al. [16] applied RL methods to optimize subtree selection and node splitting when building an R-tree. For traffic signal control, Wei et al. [41] reviewed existing RL-based approaches used in this domain.

In recent years, RL-based solutions have been investigated for trajectory analysis. For instance, Wang et al. [39] proposed an RL-based algorithm called RLS for sub-trajectory similarity search. Wang et al. [38] presented an RL-based solution called RLTS to address the trajectory simplification problem. These works differ fundamentally from ours in terms of MDP design and corresponding algorithm. For example, RLS models a state by considering the similarity between the query trajectory and the sub-trajectories, while RLTS considers the error caused by dropping a point. In contrast, our work focuses on the distance between cluster centers and trajectories, making their algorithms cannot be applied to our problem. Therefore, we do the pioneering work of proposing an RL-based algorithm for the sub-trajectory clustering problem.

7 Conclusion

In this paper, we study the sub-trajectory clustering problem and propose a novel RL-based approach. Specifically, our model is trained to consider clustering quality in guiding trajectory segmentation, enabling the segmentation and clustering components to cooperate closely and enhance each other for better clustering results. Compared with existing methods, our approach is more general and avoids developing complicated hand-crafted rules. We have conducted extensive experiments on two real-world trajectory datasets to evaluate our method. The results demonstrate that our RL-based algorithm outperforms state-of-the-art methods in terms of both effectiveness and efficiency. Some open problems for future work include: (1) How to extend this work for multi-modal trajectories, such as semantic trajectories that contain spatial, temporal, and semantic features; and (2) How to accelerate the training process.

Acknowledgements This work was supported by the NSFC (61832017), Alibaba Group through Alibaba Innovative Research (AIR) Program, the National Key Research and Development Program of China (2020YFB1710200), and Hangzhou Qianjiang Distinguished Expert Program.

References

1. Agarwal, P.K., Fox, K., Munagala, K., Nath, A., Pan, J., Taylor, E.: Subtrajectory clustering: Models and algorithms. In: SIGMOD, pp. 75–87 (2018)
2. Alt, H., Godau, M.: Computing the fréchet distance between two polygonal curves. Int. J. Comput. Geom. Appl **5**(01–02), 75–91 (1995)
3. Anagnostopoulos, A., Vlachos, M., Hadjieleftheriou, M., Keogh, E., Yu, P.S.: Global distance-based segmentation of trajectories. In: SIGKDD, pp. 34–43 (2006)
4. Ankerst, M., Breunig, M.M., Kriegel, H.P., Sander, J.: Optics: Ordering points to identify the clustering structure. ACM SIGMOD Rec. **28**(2), 49–60 (1999)
5. Buchin, K., Buchin, M., Gudmundsson, J., Löffler, M., Luo, J.: Detecting commuting patterns by clustering subtrajectories. Int. J. Comput. Geom. Appl. **21**(03), 253–282 (2011)
6. Buchin, M., Driemel, A., Van Kreveld, M., Sacristán, V.: Segmenting trajectories: a framework and algorithms using spatiotemporal criteria. J. Spatial Inf. Sci. **3**, 33–63 (2011)
7. Chen, L., Gao, Y., Fang, Z., Miao, X., Jensen, C.S., Guo, C.: Real-time distributed co-movement pattern detection on streaming trajectories. In: Proceedings of the VLDB Endowment (2019)
8. Dutta, S., Das, A., Patra, B.K.: Clustmosa: Clustering for gps trajectory data based on multi-objective simulated annealing to develop mobility application. Appl. Soft Comput. **130**, 109655 (2022)
9. Ester, M., Kriegel, H.P., Sander, J., Xu, X., et al.: A density-based algorithm for discovering clusters in large spatial databases with noise. In: SIGKDD, pp. 226–231 (1996)
10. 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 (2019)
11. Fang, Z., Du, Y., Chen, L., Hu, Y., Gao, Y., Chen, G.: E 2 dtc: An end to end deep trajectory clustering framework via self-training. In: ICDE, pp. 696–707 (2021)
12. Ferreira, N., Klosowski, J.T., Scheidegger, C.E., Silva, C.T.: Vector field k-means: Clustering trajectories by fitting multiple vector fields. In: Computer Graphics Forum, vol. 32, pp. 201–210. Wiley Online Library (2013)
13. Frentzos, E., Gratsias, K., Theodoridis, Y.: Index-based most similar trajectory search. In: ICDE (2007)
14. Gaffney, S., Smyth, P.: Trajectory clustering with mixtures of regression models. In: SIGKDD, pp. 63–72 (1999)
15. Grünwald, P.D., Myung, I.J., Pitt, M.A.: Advances in minimum description length: theory and applications (2005)
16. Gu, T., Feng, K., Cong, G., Long, C., Wang, Z., Wang, S.: A reinforcement learning based r-tree for spatial data indexing in dynamic environments. arXiv preprint arXiv:2103.04541 (2021)
17. Lee, J.G., Han, J., Li, X., Gonzalez, H.: Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering. VLDB **1**(1), 1081–1094 (2008)
18. Lee, J.G., Han, J., Whang, K.Y.: Trajectory clustering: a partition-and-group framework. In: SIGMOD, pp. 593–604 (2007)
19. Li, Y., Luo, J., Chow, C.Y., Chan, K.L., Ding, Y., Zhang, F.: Growing the charging station network for electric vehicles with trajectory data analytics. In: ICDE, pp. 1376–1387 (2015)

20. Li, Z., Lee, J.G., Li, X., Han, J.: Incremental clustering for trajectories. In: International conference on database systems for advanced applications, pp. 32–46 (2010)
21. Lloyd, S.: Least squares quantization in pcm. IEEE Trans. Inf. Theory **28**(2), 129–137 (1982)
22. Marcus, R., Negi, P., Mao, H., Tatbul, N., Alizadeh, M., Kraska, T.: Bao: Making learned query optimization practical. In: SIGMOD, pp. 1275–1288 (2021)
23. Meratnia, N., et al.: Spatiotemporal compression techniques for moving point objects. In: EDBT (2004)
24. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602 (2013)
25. Pelekis, N., Tampakis, P., Vodas, M., Panagiotakis, C., Theodoridis, Y.: In-dbms sampling-based sub-trajectory clustering. In: EDBT, pp. 632–643 (2017)
26. Puterman, M.L.: Markov decision processes: discrete stochastic dynamic programming. Wiley, New York (2014)
27. Qian, W.N., Zhou, A.Y.: Analyzing popular clustering algorithms from different viewpoints. J. Softw. **13**(8), 1382–1394 (2002)
28. Qiao, D., Yang, X., Liang, Y., Hao, X.: Rapid trajectory clustering based on neighbor spatial analysis. Pattern Recogn. Lett. **156**, 167–173 (2022)
29. Schiller, P.L., Kenworthy, J.R.: An introduction to sustainable transportation: Policy, planning and implementation. Routledge (2017)
30. Soares Júnior, A., Moreno, B.N., Times, V.C., Matwin, S., Cabral, L.D.A.F.: Grasp-uts: an algorithm for unsupervised trajectory segmentation. Int. J. Geogr. Inf. Sci. **29**(1), 46–68 (2015)
31. Sutton, R.S., Barto, A.G.: Reinforcement learning: An introduction. MIT Press, Cambridge (2018)
32. Tampakis, P., Doulkeridis, C., Pelekis, N., Theodoridis, Y.: Distributed subtrajectory join on massive datasets. ACM Trans. Spatial Algorith. Syst. (TSAS) **6**(2), 1–29 (2020)
33. Tampakis, P., Pelekis, N., Doulkeridis, C., Theodoridis, Y.: Scalable distributed subtrajectory clustering. In: 2019 IEEE international conference on big data (Big Data), pp. 950–959. IEEE (2019)
34. Wang, S., Bao, Z., Culpepper, J.S., Sellis, T., Qin, X.: Fast large-scale trajectory clustering. VLDB **13**(1), 29–42 (2019)
35. Wang, S., Bao, Z., Culpepper, J.S., Sellis, T., Sanderson, M., Qin, X.: Answering top-k exemplar trajectory queries. In: ICDE, pp. 597–608 (2017)
36. Wang, S., Shen, Y., Bao, Z., Qin, X.: Intelligent traffic analytics: from monitoring to controlling. In: Proceedings of the Twelfth ACM International Conference on Web Search and Data Mining, pp. 778–781 (2019)
37. Wang, W., Xia, F., Nie, H., Chen, Z., Gong, Z., Kong, X., Wei, W.: Vehicle trajectory clustering based on dynamic representation learning of internet of vehicles. IEEE Trans. Intell. Transp. Syst. **22**(6), 3567–3576 (2020)
38. Wang, Z., Long, C., Cong, G.: Trajectory simplification with reinforcement learning. In: ICDE, pp. 684–695 (2021)
39. Wang, Z., Long, C., Cong, G., Liu, Y.: Efficient and effective similar subtrajectory search with deep reinforcement learning. VLDB **13**(12), 2312–2325 (2020)
40. Watkins, C.J., Dayan, P.: Q-learning. Mach. Learn. **8**(3), 279–292 (1992)
41. Wei, H., Zheng, G., Gayah, V., Li, Z.: Recent advances in reinforcement learning for traffic signal control: A survey of models and evaluation. ACM SIGKDD Explorat. Newsl. **22**(2), 12–18 (2021)
42. Xia, Y., Zhou, L.: Improved clustering algorithm based on hypercube. In: 2022 International Conference on Machine Learning, Control, and Robotics (MLCR), pp. 32–37 (2022)
43. Yang, Z., Chandramouli, B., Wang, C., Gehrke, J., Li, Y., Minhas, U.F., Larson, P.Å., Kossmann, D., Acharya, R.: Qd-tree: Learning data layouts for big data analytics. In: SIGMOD, pp. 193–208 (2020)
44. Yao, D., Zhang, C., Zhu, Z., Huang, J., Bi, J.: Trajectory clustering via deep representation learning. In: IJCNN, pp. 3880–3887 (2017)
45. Yi, B.K., Jagadish, H.V., Faloutsos, C.: Efficient retrieval of similar time sequences under time warping. In: ICDE, pp. 201–208 (1998)
46. Yu, X., Li, G., Chai, C., Tang, N.: Reinforcement learning with tree-lstm for join order selection. In: ICDE, pp. 1297–1308 (2020)
47. Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G., Huang, Y.: T-drive: driving directions based on taxi trajectories. In: SIGSPATIAL, pp. 99–108 (2010)
48. Zhang, D., Chang, Z., Yang, D., Li, D., Tan, K.L., Chen, K., Chen, G.: Squid: subtrajectory query in trillion-scale gps database. VLDB J. pp. 1–18 (2023)
49. Zhang, T., Ramakrishnan, R., Livny, M.: Birch: an efficient data clustering method for very large databases. ACM SIGMOD Rec. **25**(2), 103–114 (1996)
50. Zhang, X., Meng, F., Xu, J.: Perfinsight: A robust clustering-based abnormal behavior detection system for large-scale cloud. In: IEEE CLOUD, pp. 896–899 (2018)
51. 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)
52. Zygouras, N., Gunopulos, D.: Corridor learning using individual trajectories. In: MDM, pp. 155–160 (2018)