Efficient Trajectory Compression and Range Query Processing

Hongbo Yin[*] · ✉ Hong Gao[*] · Binghao Wang[*] · Sirui Li[*] · Jianzhong Li[*]

Abstract Nowadays, there are ubiquitousness of GPS sensors in various devices collecting, transmitting and storing tremendous trajectory data. However, such an unprecedented scale of GPS data has posed an urgent demand for not only an effective storage mechanism but also an efficient query mechanism. Line simplification in online mode, searving as a mainstream trajectory compression method, plays an important role to attack this issue. But for the existing algorithms, either their time cost is extremely high, or the accuracy loss after the compression is completely unacceptable. To attack this issue, we propose ϵ_Region based Online trajectory Compression with Error bounded (ROCE for short), which makes the best balance among the accuracy loss, the time cost and the compression rate. The range query serves as a primitive, yet quite essential operation on analyzing trajectories. Each trajectory is usually

Hongbo Yin
hongboyin@hit.edu.cn

✉ Hong Gao, Corresponding author
honggao@hit.edu.cn

Binghao Wang
wangbinghao@hit.edu.cn

Sirui Li
kuwylsr@hit.edu.cn

Jianzhong Li
lijzh@hit.edu.cn

[*]School of Computer Science and Technology, Harbin Institute of Technology, Harbin 150001, China

seen as a sequence of discrete points, and in most previous work, a trajectory is judged to be overlapped with the query region R iff there is at least one point in this trajectory falling in R. But this traditional criteria is not suitable when the queried trajectories are compressed, because there may be hundreds of points discarded between each two adjacent points and the points in each compressed trajectory are quite sparse. And many trajectories could be missing in the result set. To address this, in this paper, a new criteria based on the probability and an efficient Range Query processing algorithm on Compressed trajectories RQC are proposed. In addition, an efficient index *ASP_tree* and lots of novel techniques are also presented to accelerate the processing of trajectory compression and range queries obviously. Extensive experiments have been done on multiple real datasets, and the results demonstrate superior performance of our methods.

Keywords trajectory compression · range query · compressed trajectories · accuracy loss metric

1 Introduction

With the unprecedented growth of GPS-equipped devices, such as smartphones, vehicles and wearable smart devices, massive and increasing volumes of trajectories recording the movements of humans, vehicles, or animals, are being generated for location based services, trajectory mining, wildlife tracking or other useful applications. For example, DiDi Chuxing is China's largest online ridesharing platform. It needs to process up to fifty million trip requests in a single day[1], i.e., up to thousands of requests in a rush second. This also suggests that thousands or even tens of thousands of trajectories are generated per second. However, such an increasing amount of the trajectory data collected brings a great deal of hardship on not only storing but also querying.

As an effective solution to solve the problem, line simplification, a mainstream lossy trajectory compression method, uses a sequence of consecutive line segments with much smaller size to approximately represent the trajectories and has drawn wide attention. The existing line simplification methods fall into two categories, i.e. batch mode and online mode. For each trajectory, algorithms in batch mode, such as Douglas-Peucker[9], SP[5], Intersect[5] and Error-Search[22], require that all points in this trajectory must be loaded in the local buffer before compression, which means that the local buffer must be large enough to hold the entire trajectory at least. Thus, the space complexities of these algorithms are at least $O(N)$, or even $O(N^2)$, where N is the number of input trajectory points. Such high space complexities limit the application of these algorithms in resource-constrained environments, such as the tiny tracking devices on flying foxes, whose RAM barely reaches 4 KBytes[19]. Therefore, more work focuses on the other kind of compression methods, algorithms in online mode, which only need a limited and quite small size of

---

[1]   https://tech.sina.com.cn/roll/2020-08-26/doc-iivhvpwy3125825.shtml

Table 1 The time cost and accuracy loss of some compression algorithms in online mode

Compression Algorithm	BQS[19, 20]	FBQS[19, 20]	Angular[15]	Interval[16]	OPERB[18]
Execution Time per Point (μs)	500.91	405.38	0.20	0.28	0.97
The Maximum PED Error	38.23	36.63	1532.65	1889.81	306.20

local buffer to compress trajectories in an online processing manner. Thus there are much more application scenarios where algorithms in online mode can be used, such as compressing streaming data. For these algorithms, there is a tradeoff among the execution time, the accuracy loss and the compression rate, which are the three indicators used to measure their performance. And the key issue is how to reach a good balance. As reported in Table 1, part of the experimental results[33], for existing compression algorithms, either the time cost is extremely high, such as BQS and FBQS, or the accuracy loss of the compressed trajectories is totally intolerable, such as Angular, Interval and OPERB. So for algorithms in online mode, it is still a great challenge to compress trajectories with less execution time and less accuracy loss.

To attack this issue, we propose a new online line simplification compression algorithm ROCE with only $O(N)$ time complexity and $O(1)$ space complexity, which makes the best balance among the accuracy loss, the time cost and the compression rate. Among the fastest algorithms, the accuracy loss of the compressed trajectories generated by ROCE is always the smallest, and among algorithms with the smallest accuracy loss, ROCE is always the fastest.

Compressing trajectories can reduce not only the cost of storage and transmission, but also the cost of queries greatly. Large trajectory data facilitates various real-world applications, such as trajectory pattern mining, route planing and travel time prediction. For these various applications, there is a type of trajectory queries named range queries, serving as a primitive, yet essential operation. The previous work related to range queries, such as [8, 32, 33], usually see each trajectory as a sequence of discrete points, and a trajectory is regarded to be overlapped with the query region R iff there exist one point in this trajectory falls in R. However, this traditional criteria is completely unsuitable for range quering on compressed trajectories, and many trajectories will be missing in the result set. Because there may be hundreds of points discarded between each two adjacent points in compressed trajectories and the points in each compressed trajectory are extremely sparse. If some points in a trajectory fall in the query region, but these points are discarded after the compression, such as the situation shown in Figure 1, then in the final result set of the range query, such a trajectory is missing. To solve this problem, we propose a specially designed criteria about range queries on compressed trajectories and an effective algorithm about how to process range queries on compressed trajectories with just a little additional information. And the difference between the range query result on compressed trajectories and that on the corresponding raw trajectories can be reduced greatly.

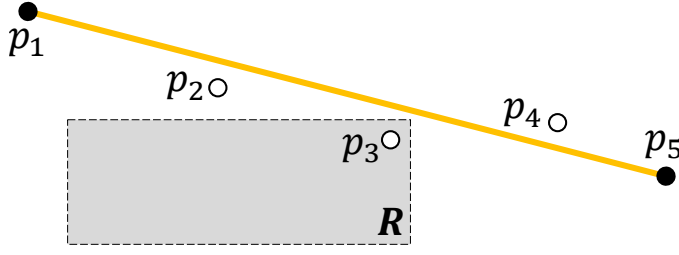The main contributions of our work are listed as follows:

Fig. 1 The discarded points p_3 falls in query region R

- Point-to-Segment Euclidean Distance (PSED), a more reasonable accuracy loss metric, is defined to measure the degree of the accuracy loss after a trajectory is compressed.
- Based on PSED, we propose a new online line simplification compression algorithm ROCE with only $O(N)$ time complexity and $O(1)$ space complexity, which achieves the best balance among the accuracy loss, the time cost and the compression rate.
- For range queries on compressed trajectories, a new criteria based on the probability and a new range query processing algorithm RQC is proposed to reduce the difference between the query results on compressed trajectories and on the corresponding raw trajectories greatly. An efficient index *ASP_tree* is also presented to accelerate the processing of range queries greatly.
- Extensive comparison experiments were conducted on real-life trajectory datasets, and the results demonstrate the superior performance of our methods.

The rest of this paper is organized as follows. We present a new accuracy loss metric PSED and the compression algorithm ROCE in Section 2. The index *ASP_tree* and the range query processing algorithm RQC are introduced in Section 3. Section 4 shows the detailed experimental results and the corresponding analysis. Section 5 reviews the related works, and Section 6 concludes our work.

2 ROCE Compression Algorithm

In this section, a more reasonable accuracy loss metric PSED is proposed first. Then based on PSED, a new compression algorithm ROCE, which makes the best balance among the accuracy loss, the time cost and the compression rate, is introduced in detail.

2.1 Basic Concepts and Notations

Definition 1 (Trajectory T): A trajectory T can be expressed as a sequence of discrete points $\{p_1, p_2, ..., p_N\}$, where $T[i] = p_i(x, y, t)$ means that the moving

object was located at longitude x and latitude y at time t. And $\forall 1 \leq i \leq j \leq N$, $p_i.t < p_j.t$.

Given a trajectory $T = \{p_1, p_2, ..., p_N\}$, $\forall i, j (1 \leq i < j \leq N)$, $T[i : j] = \{p_i, p_{i+1}, ..., p_j\}$ represents a trajectory segment with $(j - i + 1)$ consecutive points. And the line segment $p_i p_j$ can approximately represent such a trajectory segment, i.e., $p_i p_j$ is the compressed form of $T[i : j]$. p_{i+1}. p_{i+2}, ..., p_{j-1} are called the discarded points, and $p_i p_j$ is called the corresponding line segments of p_i, p_{i+1}, ..., p_j.

For a trajectory T, a compression algorithm is to divide T into a sequence of consecutive trajectory segments $\{T[i_1 : i_2], T[i_2 : i_3], ..., T[i_{n-1} : i_n]\}(i_1 = 1, i_n = N)$, and each trajectory segment $T[i_k : i_{k+1}]$ is approximately represented by the line segment $p_{i_k} p_{i_{k+1}}$. Then the corresponding compressed trajectory T' of T consists of a sequence of $n - 1$ consecutive line segments $p_{i_1} p_{i_2}, p_{i_2} p_{i_3}, ..., p_{i_{n-1}} p_{i_n}$, and T' is denoted as $\{p_{i_1}, p_{i_2}, ..., p_{i_n}\}(i_1 = 1, i_n = N)$ to simplify the representation. These consecutive line segments approximately describe the movement of the moving object. In order to distinguish an uncompressed trajectory from its corresponding compressed trajectory, we call it a raw trajectory in the following.

Definition 2 (Compression Rate): Given a compressed trajectory, $T' = \{p_{i_1}, p_{i_2}, ..., p_{i_n}\}(i_1 = 1, i_n = N)$ with $n - 1$ consecutive line segments, and its corresponding raw trajectory $T = \{p_1, p_2, ..., p_N\}$ with N points, the compression rate is defined as:

$$r = \frac{N}{n}.$$

2.2 Accuracy Loss Metric

After compression, a set of consecutive line segments is used to approximately represent a raw trajectory. When the compression rate is fixed, for a compression algorithm, the smaller accuracy loss of compressed trajectories, the better. And how to measure the accuracy loss calls for a reasonable enough metric. Usually, the accuracy loss of a compressed trajectory is calculated based on the deviation between each discarded point and its corresponding line segment.

Perpendicular Euclidean Distance (PED for short), an accuracy loss metric adopted by most line simplification methods[18], e.g. OPW[17], OPW-TR[23], BQS[19, 20] and OPERB[18], is formally defined as:

Definition 3 (PED): Given a trajectory segment $T[s : e](s < e)$ and the line segment $p_s p_e$, the compressed form of $T[s : e](s < e)$, for any discarded point $p_m(s < m < e)$ in $T[s : e]$, the PED of p_m is calculated as:

$$PED(p_m) = \frac{||\overrightarrow{p_s p_m} \times \overrightarrow{p_s p_e}||}{||\overrightarrow{p_s p_e}||}$$

where $\times$ and $|| \ ||$ are respectively to calculate the results of cross product and the length of a vector.
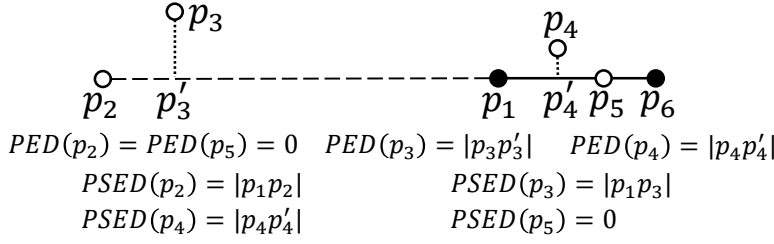
Fig. 2 An example shows how to calculate PED and PSED

PED measures the deviation between each discarded point and its corresponding line segment by using the shortest Euclidean distance from the discarded point to the straight line on which the corresponding line segment lies. However, PED can hardly describe the deviation accurately when the moving direction changes sharply. For example, it is a quite common situation that active tracked animals or wandering tourists always change their moving direction sharply. Figure 2 illustrates the tracked object makes a U-turn, and the line segment p_1p_6 approximately represents the trajectory segment $T[1:6]$ after the compression. The accuracy losses of p_2 and p_3 in PED are respectively 0 and $|p_3p_3'|$. But in fact, p_2 is obviously far away from the line segment p_1p_6 and the distance between p_3 and the p_1p_6 is also far more than $|p_3p_3'|$. The reason for these is that $PED(p_2)$ and $PED(p_3)$ are both calculated based on the perpendicular distance between the discarded points and the extension line of p_1p_6. Thus the compressed trajectories, which are generated by the compression algorithms whose accuracy loss metric is PED, are not able to reflect the real movement patterns.

To attack this issue, we define a more reasonable accuracy loss metric PSED to measure the accuracy loss after the compression. The key difference between PSED and PED is that PSED adopts the shortest Euclidean distance from a point to its corresponding line segment, rather than the straight line on which the corresponding line segment lies. PSED is formally defined as follows:

Definition 4 (PSED): Given a trajectory segment $T[s:e](s<e)$ and the line segment p_sp_e, the compressed form of $T[s:e](s<e)$, for any discarded point $p_m(s<m<e)$ in $T[s:e]$, the PSED of p_m is calculated according to the following cases:

$$PSED(p_m) = \begin{cases} \frac{||\overrightarrow{p_sp_m} \times \overrightarrow{p_sp_e}||}{||\overrightarrow{p_sp_e}||}, & \overrightarrow{p_sp_m} \cdot \overrightarrow{p_sp_e} \geq 0 \text{ and } \overrightarrow{p_mp_e} \cdot \overrightarrow{p_sp_e} \geq 0 \\ min(||\overrightarrow{p_sp_m}||, ||\overrightarrow{p_mp_e}||), & \text{otherwise} \end{cases}$$

where $\times$ and $\cdot$ are respectively to calculate the results of cross product and dot product.

In Definition 4, that both $\overrightarrow{p_sp_m} \cdot \overrightarrow{p_sp_e} \geq 0$ and $\overrightarrow{p_mp_e} \cdot \overrightarrow{p_sp_e} \geq 0$ are satisfied means that the perpendicular point of p_m falls on the line segment p_sp_e. In Figure 2, since the perpendicular points of p_2 and p_3 both fall on

the extension line of p_1p_6, $PSED(p_2) = min(|p_1p_2|, |p_2p_6|) = |p_1p_2|$ and $PSED(p_3) = min(|p_1p_3|, |p_3p_6|) = |p_1p_3|$. For p_4 and p_5, whose corresponding perpendicular points are both on the line segment p_1p_6, $PSED(p_4) = |p_4p_4'|$ and $PSED(p_5) = 0$.

Based on PSED, the ϵ-error-bounded compressed trajectory is defined as follows:

Definition 5 (ϵ-error-bounded Compressed Trajectory): Given a threshold value ϵ, a compressed trajectory $T' = \{p_{i_1}, p_{i_2}, ..., p_{i_n}\}(p_{i_1} = p_1, p_{i_n} = p_N)$ and its corresponding raw trajectory $T = \{p_1, p_2, ..., p_N\}$. If $\forall p_m \in T, PSED(p_m) \leq \epsilon$, then T' is ϵ-error-bounded, and ϵ is an upper bound of PSED.

2.3 Algorithm ROCE

In this part, a new compression algorithm in online mode named ROCE, which makes the best balance among the accuracy loss, the time cost and the compression rate, is presented. Given a raw trajectory $T = \{p_1, p_2, ..., p_N\}$ and the upper bound of PSED ϵ, by adopting a greedy strategy, ROCE is to compress T into an ϵ-error-bounded compressed trajectory T', which consists of a sequence of consecutive line segments.

In order to determine whether a compressed trajectory is ϵ-error-bounded or not much more conveniently, we define a new concept ϵ_Region as below:

Definition 6 (ϵ_Region C_i): Given a raw trajectory point p_i and the upper bound of PSED ϵ, the circle region C_i, whose center and radius are respectively p_i and ϵ, is called the ϵ_Region of p_i.

Then, it is quite easy to get the following property about ϵ_Region:

Lemma 1 *A trajectory segment $T[s : e](s < e)$ is compressed into a line segment p_sp_e. For any discarded point $p_m(s < m < e)$ in $T[s : e]$, $PSED(p_m) \leq \epsilon$, where ϵ is the upper bound of PSED, iff p_sp_e intersects C_m. p_sp_e is ϵ-error-bounded iff p_sp_e intersects all ϵ_Regions of discarded points, i.e. C_{s+1}, C_{s+2}, ..., C_{e-1}.*

As shown in Figure 3, the raw trajectory $T = \{p_1, p_2, ..., p_8\}$ is compressed into T', which consists of two line segments, i.e. p_1p_6 and p_6p_8. For any discarded point in the trajectory segment $T[1 : 6]$, p_1p_6 intersects its corresponding ϵ_Region. Thus, p_1p_6 is clearly ϵ-error-bounded. It is obvious that the line segment p_6p_8 does not intersect the corresponding ϵ_Region of p_7, i.e. C_7, and $PSED(p_7) > \epsilon$. Thus neither p_6p_8 nor T' is ϵ-error-bounded.

Given the upper bound of PSED ϵ and a raw trajectory $T = \{p_1, p_2, ..., p_N\}$, an optimal compression is to compress T into an ϵ-error-bounded trajectory T' consisting of the smallest number of consecutive line segments. T can be divided into 2^{N-1} different sets of consecutive trajectory segments, which means that there are up to 2^{N-1} different compressed strategies and the search space is exponential. By adopting a greedy strategy and some effective tricks, ROCE,
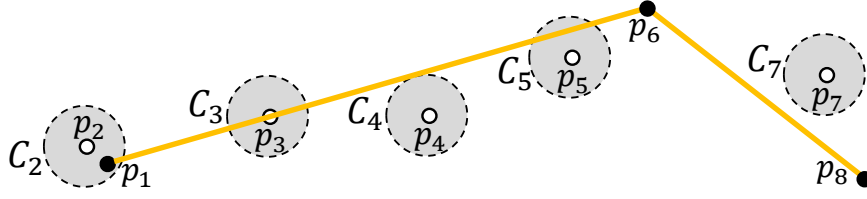
Fig. 3 $T' = \{p_1, p_6, p_8\}$ is a compressed trajectory of $T = \{p_1, p_2, ..., p_8\}$

an efficient approximate algorithm, compresses trajectories in an online processing manner. The first thing, ROCE anchors the start point p_s of a trajectory segment to be compressed. p_f, where f is a variable and initialized to $(s + 2)$, is selected as the current float point. Then by using p_s and p_f a trajectory segment $T[s : f]$ is defined. p_{f+1} is assigned as the new float point and $(f + 1)$ is assigned as f, if $\forall p_m(s < m < f) \in T[s : f]$, $PSED(p_m) \leq \epsilon$. Otherwise, $T[s : f - 1]$ is compressed into a line segment $p_s p_{f-1}$, and the anchor point of the next trajectory segment to be compressed is set to p_{f-1}.

Each time ROCE checks whether the last float point p_{f-1} is the final end point of the current trajectory segment or not, each point $p_m(s < m < f)$ needs to be scanned to calculate the corresponding PSED to verify whether the line segment $p_s p_f$ is ϵ-error-bounded. So each point needs to be scanned many times during the compression, and lots of execution time is wasted here. To attack this issue, the candidate region is adopted by ROCE, and each point just needs to be scanned only once. $(p_s, p_f)\_Region$ and $T[s : f]\_CandidateRegion$ are formally defined as follows:

Definition 7 ($(p_s, p_f)\_Region$): Given a trajectory segment $T[s : f](s < f$ and $|p_s p_f| > \epsilon)$ and the upper bound of PSED ϵ. Then p_s is outside the ϵ_Region C_f of p_f, and two tangent rays of C_f starting from p_s named $tr_{s,f}$ and $tr'_{s,f}$ can be gotten. The minor sector enclosed by $tr_{s,f}$ and $tr'_{s,f}$, excluding the circular region, whose center and radius are p_s and $|p_s p_f|$ respectively, is called $(p_s, p_f)\_Region$.

Definition 8 ($T[s : f]\_CandidateRegion$): Given a trajectory segment $T[s : f](s < f$ and $|p_s p_f| > \epsilon)$ and the upper bound of PSED ϵ. $T[s : f]\_CandidateRegion$ $= (p_s, p_{s+1})\_Region \bigcap (p_s, p_{s+2})\_Region \bigcap ... \bigcap (p_s, p_f)\_Region$, i.e., $T[s : f]\_CandidateRegion = T[s : f - 1]\_CandidateRegion \bigcap (p_s, p_f)\_Region$ if $s < (f - 1)$.

During the procedure of finding which is the final end point of the current trajectory segment starting from p_s to be compressed, if the float point p_{f+1} falls in $T[s : f]\_CandidateRegion$, then $\forall p_m(s < m < f + 1)$, $PSED(p_m) \leq \epsilon$. So by using the candidate region in ROCE, PSED no longer needs to be calculated any more, and each point just needs to be scanned only once to update the current candidate region. Figure 4 gives us an example to show how to update the candidate region. Since p_1 is outside the ϵ_Region C_2 of p_2, we can get two tangent rays $tr_{1,2}$ and $tr'_{1,2}$ of C_2. Both $(p_1, p_2)\_Region$ and $T[1 :$
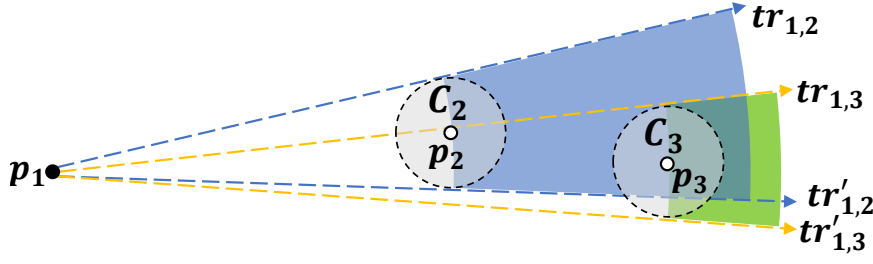
Fig. 4 An example shows how to update the candidate region

$2]\_CandidateRegion$ are the region in blue. p_3 falls in $T[1:2]\_CandidateRegion$. Similarly, $(p_1, p_3)\_Region$ is the region in green. Then $T[1:3]\_CandidateRegion$ is the overlapping region of $T[1:2]\_CandidateRegion$ and $(p_1, p_3)\_Region$. According to Lemma 1, the line segment $p_1 p_4$ is ϵ-error-bounded iff the next point p_4 falls in $T[1:3]\_CandidateRegion$, because the line segment $p_1 p_4$ must intersect all $\epsilon\_Regions$ of discarded points, i.e., C_2 and C_3.

The pseudo code of ROCE is formally introduced in Algorithm 1. Starting from the first point, points in the trajectory are scanned one by one. In each iteration, ROCE tries to find which is the final end point of the current trajectory segment to be compressed, and this trajectory segment is compressed into a line segment by ROCE (Line 3-12). For the following points of the start point, if none of their distances to the start point are more than ϵ, for any line segment starting from the start point, it must intersect all their corresponding $\epsilon\_Regions$. Thus according to Lemma 1, their restrictions no longer need to be thought about (Line 6-7).

Algorithm 1 : ROCE Algorithm

Input: a raw trajectory $T = \{p_1, p_2, ..., p_N\}$ and the upper bound of PSED ϵ
Output: an ϵ-error-bounded compressed trajectory T' of T

1: $i \leftarrow 1, T' \leftarrow [T[1]]$
2: **while** $i \leq N$ **do**
3:     $StartPoint \leftarrow T[i]$
4:     $CandidateRegion.initialize(StartPoint)$
5:     $i \leftarrow i + 1$
6:     **while** $(dist(StartPoint, T[i]) \leq \epsilon)$ and $(i \leq N)$ **do**
7:         $i \leftarrow i + 1$
8:     **while** $(T[i]$ in $CandidateRegion)$ and $(i \leq N)$ **do**
9:         $CandidateRegion.update(T[i], \epsilon)$
10:        $i \leftarrow i + 1$
11:     $i \leftarrow i - 1$
12:     $T'.append(T[i])$
    return T'

By using the candidate region, ROCE is a one-pass error bounded trajectory compression algorithm, since each point just needs to be scanned onle once to update the current candidate region. So the time complexity of ROCE
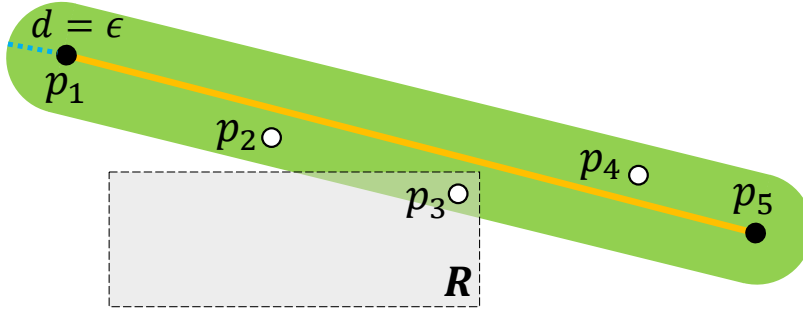
Fig. 5 The trajectory segment $T[1:5]$ is compressed into a line segment $p_1 p_5$

is $O(N)$. The space complexity of ROCE is only $O(1)$, since only constant and small space is needed by ROCE, no matter how many points to be compressed into a line segment.

3 Range Query Processing

In most previous work[8, 32, 33], each raw trajectory is usually regarded as a sequence of discrete points, and a raw trajectory is determined to be overlapped with the query region R iff at least one point in this trajectory falls in R. But this traditional criteria is not suitable for range queries on compressed trajectories, since it will lead to many trajectories are missing in the result set as discussed in Section 1. To address this, we propose a new criteria based on the probability about range queries on compressed trajectories and an effective algorithm about how to process range queries on compressed trajectories. The new criteria is formally defined as:

Definition 9 (Range Query on Compressed Trajectories): Given a query region R, a compressed trajectory dataset $\mathbb{T}'$ and a probability threshold value p, the range query result $Q_r(R, \mathbb{T}', p)$ consists of all such compressed trajectories in $\mathbb{T}'$, the probabilities P of whose corresponding raw trajectories are overlapped with R are all larger than p, i.e.

$$Q_r(R, \mathbb{T}', p) = \{T' \in \mathbb{T}' | P(\exists p_i \in RawTrajectory(T'), \ s.t. \ \ p_i \in R) > p\}$$

Though query regions are considered as two-dimensional rectangles for simplicity, our method can be easily adapted to handling query regions in arbitrary shapes. As shown in Figure 5, the trajectory segment $T[1:5]$ is compressed into a line segment $p_1 p_5$ by ROCE algorithm with the upper bound of PSED ϵ. The only certainty is that there are 3 points discarded between p_1 and p_5, and these 3 discarded points are all within the green region, named $\epsilon\_Bounding\ Region$ (ϵ_BR for short), which is formally defined as:

Definition 10 ($\epsilon\_BR(p_i p_j)$): Given a line segment $p_i p_j$ and the upper bound of PSED ϵ, $\epsilon\_BR(p_i p_j)$ is the region consists of all points, whose PSEDs to the line segment $p_i p_j$ are all less than or equal to ϵ.

For a compressed trajectories, based on the positional relationship between the ϵ_BR of each line segment and the query region R, the probability of the corresponding raw trajectories overlapped with R can be calculated, which will be introduced more detailedly in Section 3.2.

However, there are multiple consecutive line segments in each compressed trajectory, and for a range query, it will cost too much to judge the relationship between the query region R and the ϵ_BR of each line segment in all compressed trajectories. To address this, we find that for a line segment after compression, if its corresponding ϵ_BR and the query region R are not overlapped, then any discarded point approximately represented by this line segment must not fall in R. And based on this, RQC follows a filtering-and-verification framework. The filtering step can prune most invalid trajectories at quite low computation cost. In Section 3.1, Adaptive Spatial Partition quadtree like index (ASP_tree for short), a high efficient index, is proposed to accelerate the filtering step greatly. Then in Section 3.2, the processing procedure of RQC is described in detail.

3.1 Trajectory Index ASP_tree

The root node of ASP_tree represents all compressed trajectories falling in the entire region. If there are more than ξ endpoints of all line segments falling in the corresponding region of each node in ASP_tree, where ξ is a threshold value estimated through experiments, then this node is a non-leaf node with 4 child nodes. Otherwise, this node serves as a leaf node. So ξ controls the height of ASP_tree.

To reduce the space overhead of ASP_tree, the detailed information of compressed trajectories is only stored in leaf nodes in the form of $ChildRegion\_ChildPointer$, where $ChildRegion\_ChildPointer$ refers to the corresponding regions and addresses of its 4 child nodes. Each leaf node in ASP_tree stores information in the form of $ID\_LineSegments$. $ID\_LineSegments$ refers to some consecutive line segments of a compressed trajectory whose identifier is ID, and the corresponding ϵ_BRs of these line segments are all overlapped with the corresponding region of this leaf node. So for a compressed trajectory, it may be split into multiple sets of consecutive line segments and stored in different leaf nodes.

In a traditional quadtree, if a node is a father node with 4 child nodes, then the corresponding region represented by the father node is evenly divided into four disjoint regions, which are respectively assigned to these 4 child nodes. But this may make the index inclined greatly, which affects the efficiency of the range query processing, because trajectories are not evenly distributed. Thus, it is not suitable to do so. To attack this issue, a data adaptive strategy is adopted in ASP_tree. As shown in Figure 6, there are totally two ways to divide the corresponding region of a father node. For line segments whose corresponding ϵ_BRs are overlapped with this region, we first get all endpoints of these line segments falling in this region, and then get the median of all

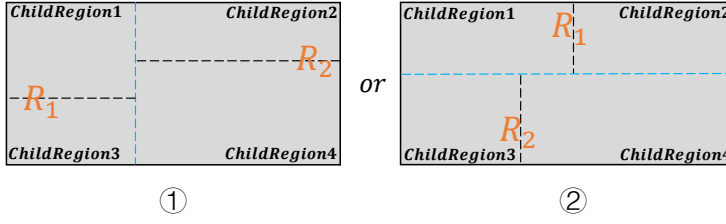**Fig. 6** An example shows how to divide the corresponding region of a father node among its 4 child nodes
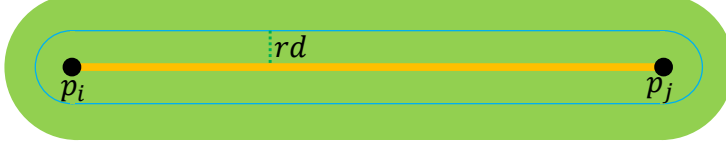

Fig. 7 The trajectory segment $T[i:j]$ is compressed into a line segment $p_i p_j$

their x dimensions (y dimensions). The median is used to draw a vertical (horizontal) line, which divides this region into two regions named R_1 and R_2. After that, the medians of all y dimensions (x dimensions) of all endpoints falling in R_1 and R_2 are respectively used to further divide these two regions into four smaller disjoint regions. For these two ways, the way with fewer repeated line segments whose corresponding ϵ_BRs are overlapped with the four smaller regions will be chosen. The purpose of doing these is to make ASP_tree balanced, which is verified by the experimental results on real-life compressed trajectories in Section 4.

When a region is to be divided into 2 disjoint regions, there may be a special case. For all line segments whose corresponding ϵ_BRs are overlapped with this region, none of their endpoints fall in this region. In such a case, this region will be divided evenly into 2 smaller regions.

3.2 Range Query Processing Algorithm

To answer a range query on compressed trajectories, the essential question is how to calculate the probablity of that at least one point in the corresponding raw trajectory of a compressed trajectory falls in the given query region R. For each line segment of compressed trajectories, the additional information we can get is how many points are discarded between the two endpoints and that these discarded points are all within the corresponding ϵ_BR of this line segment. First of all, we should know what is the probability of that there exists at least one discarded point of its corresponding line segment falls in R.

To address this, we propose Algorithm 2. After our analysis on lots of real-life compressed trajectories, it is quite clear that for a line segment of a compressed trajectory, its corresponding discarded points are not uniformly distributed in their corresponding ϵ_BR, but gather near this line segment.

Algorithm 2 : Calculating the probablity

Input: a line segment $p_i p_j$, the query region R, the upper bound of PSED ϵ, the standard variance σ of the Gaussian distribution, the number of sampling points n_s, the discarded point number n_d between p_i and p_j

Output: the probablity of that at least one discarded point of its corresponding line segment $p_i p_j$ falls in R

1: $PointsNumInR \leftarrow 0$
2: **for** $loop \leftarrow 1; loop \leq n_s; loop \leftarrow loop + 1$ **do**
3:     **do**
4:         $rd \leftarrow |generateGaussianDistribution(0, \sigma)|$
5:     **while** $rd > \epsilon$
6:     $SinglePoint \leftarrow samplingOnCurve(p_i, p_j, rd)$
7:     **if** $SinglePoint.isInQueryRegion(R)$ **then**
8:         $PointsNumInR \leftarrow PointsNumInR + 1$
9: $rate \leftarrow PointsNumInR/n_s$
    return $1 - (1 - rate)^{n_d}$

For example, for the compressed trajectories whose compression rate is 200, more than 90% of the PSEDs of discarded points are less than 0.5ϵ with ϵ being the upper bound of PSED. So for all compressed trajectories, we choose to use a Gaussian distribution to simulate the distributions of discarded points in all ϵ_BRs. The mean of the Gaussian distribution is set 0, and the standard variance σ can be got and saved when trajectories are being compressed. As shown in Figure 7, rd is the absolute value of the PSED randomly generated by using the Gaussian distribution, and we should make sure that $rd \leq \epsilon$ (Line 3-5). For any point on the curve in blue, its PSED to the line segment $p_i p_j$ is rd. On this curve, a point is randomly selected as a representative of discarded points (Line 6). After n_s points are sampled, the probability of that there exists at least one discarded point of its corresponding line segment falls in R can be estimated.

Then based on Algorithm 2, Algorithm 3 shows the pseudo code of Range Query processing algorithm on Compressed trajectories RQC. To get the range query result set $Q_r(R, \mathbb{T}', p)$, RQC is performed in four steps:

First (Line 1-8), traverse the index ASP_tree, and only all $ID\_LineSegments$ stored in leaf nodes, whose corresponding regions are overlapped with the query region, are left. So by doing this, RQC prunes most invalid compressed trajectories to form a candidate set S_c, which consists of multiple sequences of consecutive line segments.

Second (Line 9), some efficient pruning strategies based on the MBR (short for the Minimal Bounding Rectangle) are utilized to further reduce the size of the candidate set S_c, and S_{c1}, a much smaller candidate set, and S_{f1}, a part of the final result set consisting of multiple IDs of compressed trajectories, can be gotten. For a sequence of consecutive line segments in a compressed trajectory, its corresponding MBR is the smallest rectangle which contains all ϵ_BRs of these line segments. It is clear that if the MBR and the query region R do not overlap, the corresponding points in the corresponding raw trajectory must not fall in R. And if the MBR is completely contained in

---

Algorithm 3 : RQC Algorithm

Input: the query region R, the compressed trajectory dataset $\mathbb{T}'$, ASP_tree of $\mathbb{T}'$ and the
probability threshold value p
Output: the range query result set $Q_r(R, \mathbb{T}', p)$
1: $PQ \leftarrow createPriorityQueue()$
2: $PQ.push(ASP\_tree.root.FourChildNodes())$
3: **while** $PQ \neq \phi$ **do**
4:     $Node \leftarrow PQ.pop()$
5:     **if** $Node.isOverlap(R)$ and $!(Node.isLeaf())$ **then**
6:         $PQ.push(Node.FourChildNodes())$
7:     **if** $Node.isOverlap(R)$ and $Node.isLeaf()$ **then**
8:         $S_c.append(Node.allID\_LineSegments())$
9: $(S_{f1}, S_{c1}) \leftarrow S_c.PruningBasedOnMBR()$
10: $(S_{f2}, S_{c2}) \leftarrow S_{c1}.verifyByEndpoints()$
11: $S_{f3} \leftarrow S_{c2}.verifyBy\epsilon\_BRs(p)$
        return $(merge(S_{f1}, S_{f2}, S_{f3})).getTrajectories(\mathbb{T}')$

R, then there must be at least a point in the corresponding raw trajectory
falling in R, and this corresponding compressed trajectory must be in the final
result set of this range query. By using these two properties, most relationships
between sequences of consecutive line segments and the query region R can
be determined one by one.

Third (Line 10), for each sequence of consecutive line segments in the
candidate set S_{c1}, whether there exist an endpoint falling in the query region
R is determined one by one. And if the answer is yes, then the corresponding
raw trajectory must be overlapped with R and the corresponding compressed
trajectories must be in the range query result set $Q_r(R, \mathbb{T}', p)$. Then the final
candidate set S_{c2} and a part of the final result set S_{f2} can be gotten.

Last (Line 11), for S_{c2}, the final candidate set, Algorithm 2 is used to
measure the probablity of that at least one corresponding discarded point of
each line segment falls in the query region R. For a compressed trajectories
$T' = \{p_{i_1}, p_{i_2}, ..., p_{i_n}\}(i_1 = 1, i_n = N)$, supposing that the probablities of its
line segments are r_1, r_2, ..., r_{n-1} respectively, then $P(T')$, the probablity of
the corresponding raw trajectories overlapped with the query region R, can
be calculated as:

$$P(T') = 1 - \prod_{i=1}^{n-1}(1 - r_i)$$

And if $P(T') > p$ where p is the given probability threshold value, then the
ID of T' is put in the final result set S_{f3}

4 Experimental Evalution

In this section, the performances of our compression algorithm ROCE, and
range query processing algorithm RQC on compressed trajectories are evaluate
in detail.

4.1 Experiment Setup

4.1.1 Datasets

The experiments were conducted on three real-life datasets. The dataset named Animal[2][12] records the migrations of 8 young white storks originating from 8 different populations. Because of the tiny tracking devices on these young whites, the sampling rates of these trajectories are relatively low. The dataset named Indoor[3][2] records the trajectories of visitors in a shopping center, and the points were sampled very frequently. With a quite large size, the dataset named Planet[4] consists of lots of trajectories distributed all over the globe. The movement modes of these trajectories are also pretty rich. These trajectories are sparsely distributed on the Earth, but mainly gather in a large rectangular region, which is about $3.9 * 10^5 km^2$ in area. All trajectories completely contained in such a large region were selected as a raw dataset called Planet I. The trajectories with less than 1000 points were all removed, since this dataset was to be compressed. In Planet I, there are 96279 raw trajectories and 0.3 billion points in total. The experiments in Section 4.3 were all performed on Planet I and its corresponding compressed datasets compressed by ROCE with different compression rates.

Some baseline algorithms are found to be too time-consuming to run on the entire datasets when comparing the execution time of different compression algorithms. So we had to randomly sampled some long trajectories from Animal, Indoor and Planet I. Then we got 3 subsets called Animal II, Indoor II and Planet II with 120, 90 and 47 long trajectories respectively. In these three subsets, there are all about 2 million points in total.

4.1.2 Experimental Environment

All experiments were conducted on a linux machine with 32GB memory and a 64-bit, 8-core, 3.6GHz Intel(R) Core (TM) i9-9900K CPU. All algorithms were implemented in C++ on Ubuntu 18.04. Each experiment was repeated over 3 times, and the average is reported below.

4.2 Performance Evaluation for Compression Algorithms

Our compression algorithm ROCE was compared with 4 existing compression algorithms in online mode using PED as their error metric, i.e. OPW(BOPW)[17, 23], BQS[19, 20], FBQS[19, 20] and OPERB[18]. For DOTS[3], though its error metric is LISSED but not PED, it was still compared with ROCE, because it was demonstrated to have stable superiority against other compression algorithms in online mode on some indicators[33]. The performances of these

---

compression algorithms were measured by the execution time of compression and the accuracy loss of the generated compressed trajectories.
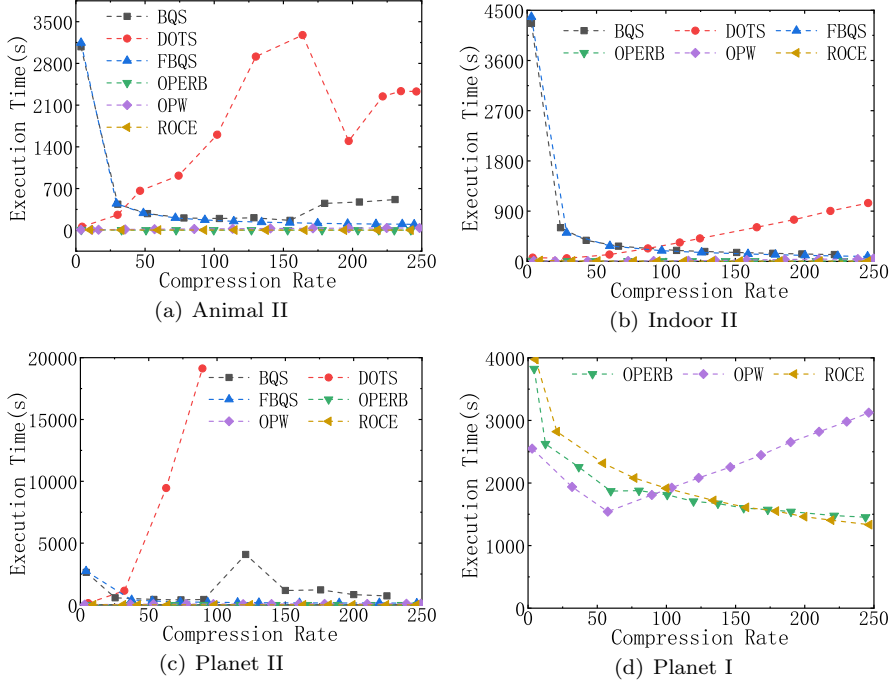

Fig. 8 Efficiency evaluation: varying the compression rate

4.2.1 Execution Time

We first evaluate the execution times of these 6 algorithms w.r.t. varying the compression rate, and the results are reported in Figure 8. ROCE, OPW and OPERB are obviously faster than BQS, FBQS and DOTS. During each loop iteration to compress a trajectory segment into a line segment, BQS and FBQS both need much more time than other algorithms at the beginning. Quite a bit of memory and time are needed by DOTS to handle the situations where the tracked object stays at the same place for a long time. And on Planet II, DOTS was too time-consuming to continue when the compression rate is close to 100, so we chose to stop the experiment. Only OPERB, OPW and ROCE were chosen to run on Planet I, since BQS, FBQS and DOTS are too slow to run on Planet I. On Planet I, the execution times of OPERB and ROCE are nearly the same, and both become shorter with the increase of the compression rate, because fewer trajectory segments need to be compressed into line segments. OPW needs more execution time with the increase of the compression rate, since the time complexity of OPW is $O(N^2)$. And when the

compression rate is more than 100, ROCE is faster than OPW. To conclude, ROCE is faster than many other compression algorithms.

To evaluate the impact of the size of each raw trajectory (i.e. the number of points in each raw trajectory) on the execution time of compression, 20 trajectories with the largest sizes were chosen from Animal, Indoor and Planet respectively, and the size of each trajectory was varied from 5000 to 20000 while the compression rates were all fixed as 50. The results are shown in Figure 9. The y-coordinates are the time rates of the execution time of compression to the one of compressing trajectories whose sizes are all 5000. Only two algorithms ROCE and OPERB always scale well with the increase of the size of each trajectory on all datasets, and show nearly linear running time. But other algorithms do not, and much more execution time is needed to compress trajectories with larger sizes, especially for DOTS.
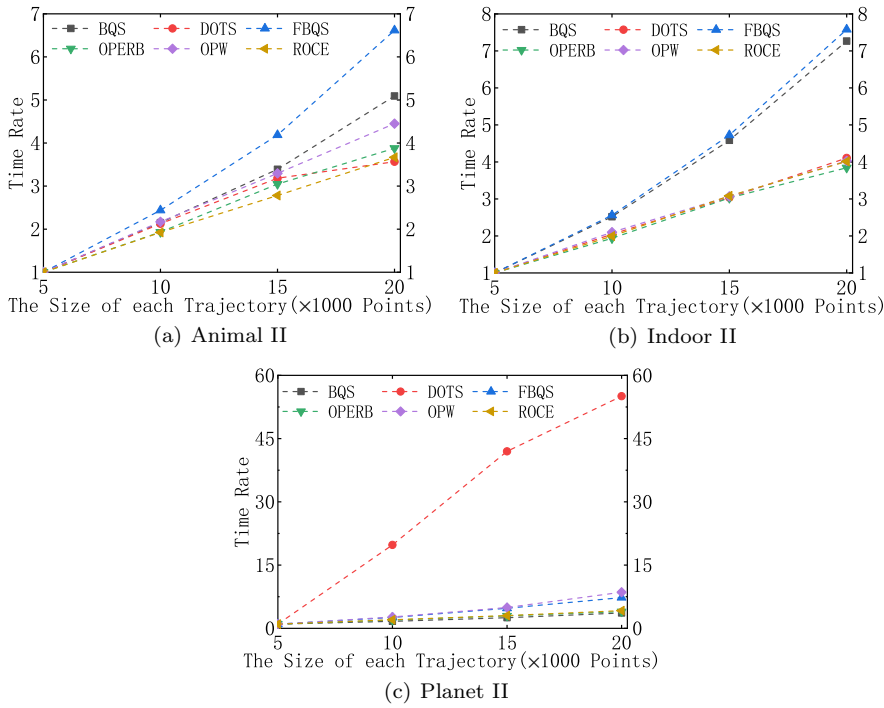

(a) Animal II

(b) Indoor II

(c) Planet II

Fig. 9 Efficiency evaluation: varying the size of trajectories

4.2.2 Accuracy Loss

For these compression algorithms, in order to compare the accuracy loss of the generated compressed trajectories the maximum and the average PSED of these compressed trajectories are evaluated w.r.t. varying the compression
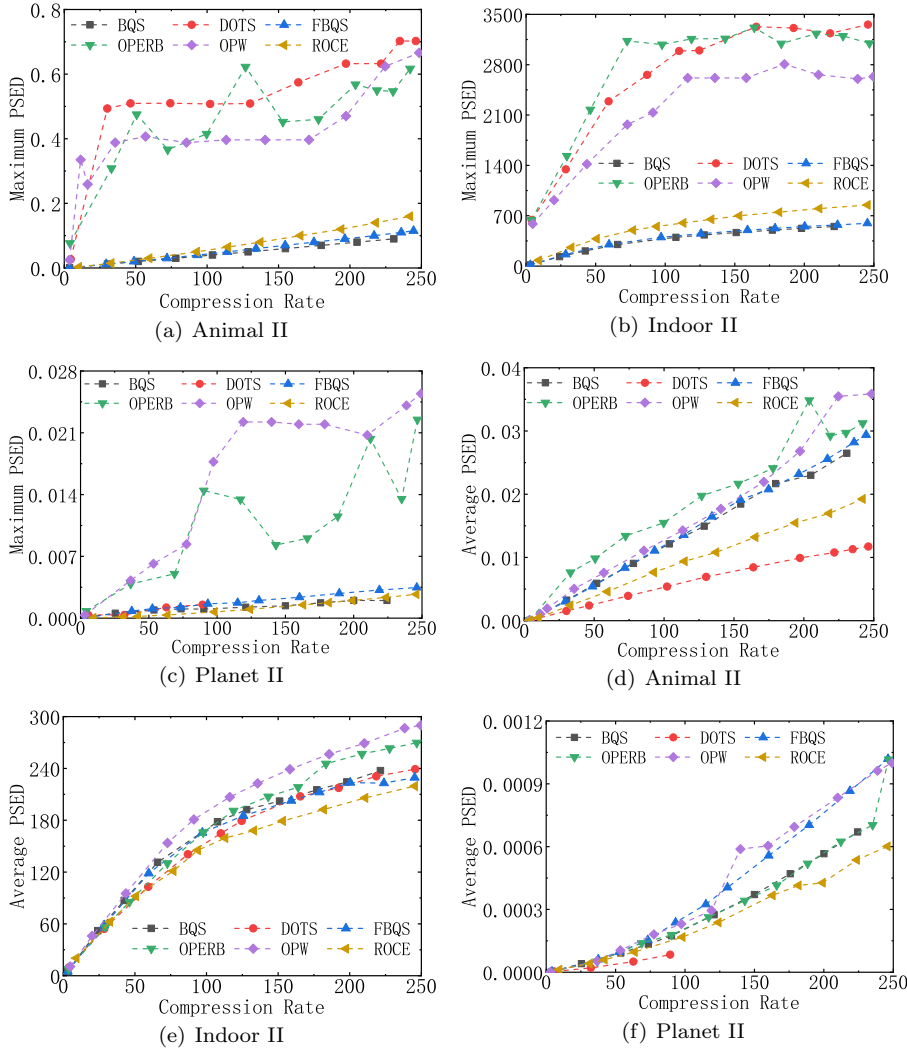
Fig. 10 Evaluation of the maximum and the average PSED: varying the compression rate

rate, and the results are reported in Figure 10. For all algorithms, both the maximum and the average PSED increase with the increase of the compression rate. In terms of execution time, ROCE performs similarly with OPERB and OPW, but ROCE always performs much better than OPERB and OPW on the maximum PSED. ROCE, BQS and FBQS always perform similarly on the maximum PSED, but both BQS and FBQS need much more execution time than ROCE. ROCE always performs much better than most other algorithms on the average PSED. So with much less exection time, the compressed trajectories generated by ROCE maintain much less accuracy loss than those

generated by most other algorithms. So, it is quite clear that ROCE makes the best balance among the accuracy loss, the time cost and the compression rate. Among the fastest algorithms, the accuracy loss of the compressed trajectories generated by ROCE is always the smallest, and among algorithms with the smallest accuracy loss, ROCE is always the fastest.

4.3 Performance Evaluation for RQC Algorithm

The range query serves as a primitive, yet essential operation. In the previous work related to range queries, such as [8, 32, 33], each trajectory is usually seen as a sequence of discrete points, and a trajectory is regarded to be overlapped with the query region R iff at least one point in this trajectory falls in R. However, this traditional criteria is completely unsuitable for range quering on compressed trajectories, and many trajectories will be missing in the result set. To solve this problem, we propose the new criteria defined in Definition 9 and the efficient RQC algorithm for range queries on compressed trajectories. On compressed trajectories, whether the traditional or the new criteria is used results in quite different range query results. In the first following experiment, we evaluate the deviation between the range query results on raw trajectories and the corresponding compressed trajectories based on different criteria. The impacts of the size of each query region, the number n_s of sampling points, the probality threshold value p, ASP_tree are also respectively studied.

4.3.1 Range Queries on Compressed Trajectories Based on the Traditional or the New Criteria

To measure the deviation between the range query results on the raw trajectories and those on the corresponding compressed trajectories, 3 evaluation metrics are defined and used here. Given a range query, the query result set on the raw trajectories based on the traditional criteria is denoted by Q_R. And Q_C represents the query result set on compressed trajectories. The precision rate Pre and the recall rate Rec of a range query result on compressed trajectories are respectively defined as:

$$Pre = \frac{|Q_R \cap Q_C|}{|Q_C|} \qquad Rec = \frac{|Q_R \cap Q_C|}{|Q_R|}$$

F_1-Measure, a comprehensive evaluation metric, is defined as:

$$\frac{1}{F_1} = \frac{1}{2} * (\frac{1}{Pre} + \frac{1}{Rec})$$

For 1000 randomly generated range queries on compressed trajectories, we evaluate the average Pre, Rec and F_1 of the query results w.r.t. varying the compression rate, and the results are shown in Figure 11. When querying on compressed trajectories based on the traditional criteria, the average Pre is

always 1, since all points in each compressed trajectory must be in the corresponding raw trajectors. But with the increase of the compression rate, Rec declines sharply, which shows that up to 45.8% of the corresponding raw trajectories overlapped with the query regions are not discovered. When querying on compressed trajectories based on the new criteria, though at most 13.4% of the corresponding raw trajectories not overlapped with the query regions appear in the result set, much more corresponding raw trajectories (i.e. at least 84.2%) overlapped with the query regions can be found out. The average F_1, the comprehensive metric of Pre and Rec, also demonstrates the stable superiority of the new criteria. In summary, it is far more suitable to answer range queries on compressed trajectories based on the new criteria.
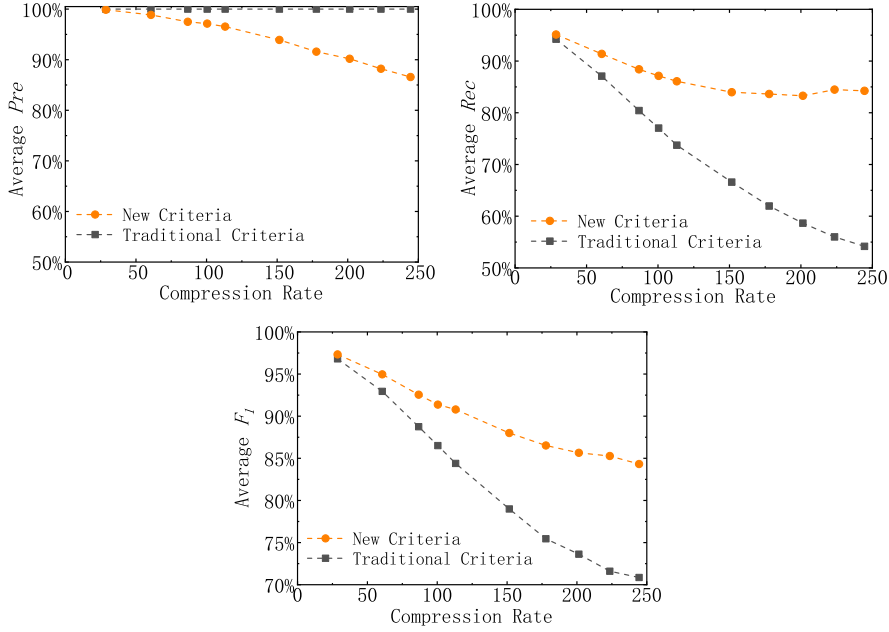

Fig. 11 Evaluation of the average Pre, Rec and F_1 of range query results on compressed trajectories: varying the compression rate

4.3.2 Impacts of the Sizes of Query Regions

For range queries, the size of the query regions may have impact on the size of range query results and the execution time. 1000 randomly generated range queries were executed on the compressed trajectories whose compression rate is 200, and the size of each query region was varied from $5km^2$ to $30km^2$ in area. The results are shown in Figure 12 and Figure 13. We can see that with the increase of the size of each query region, though the average size of query

results on compressed trajectories grows nearly linearly, the execution time needed by our RQC algorithm does not vary much. Thus RQC algorithm can easily support range queries with much larger query regions without needing more execution time. And RQC algorithm is quite efficient and a range query on more than 90000 compressed trajectories can be processed just within 2ms.
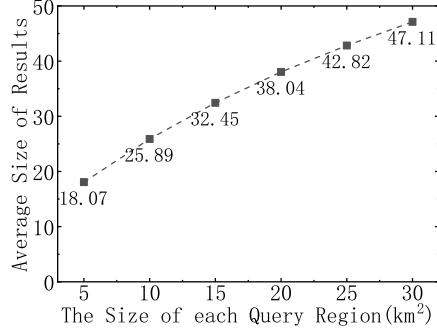

Fig. 12 Evaluation of the average size of range query results on compressed trajectories: varying the size of each query region
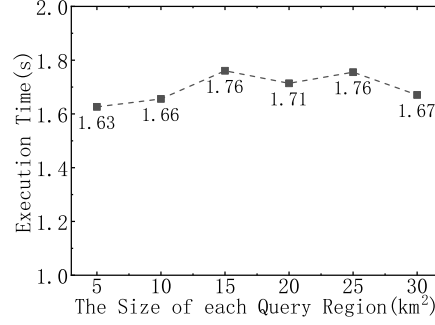
Fig. 13 Efficiency evaluation: varying the size of each query region

4.3.3 Impacts of the Number n_s of Sampling Points

To process a range query on compressed trajectories, n_s points are sampled to calculate the approximate probablity of that there exists a discarded point of its corresponding line segment falling in the query region R. We evaluate the execution time, the average precision rate Pre, the average recall rate Rec and the average F_1 of query results of 1000 randomly generated range queries on the compressed trajectories whose compression rate is 200 w.r.t. varying n_s from 10 to 1280, and the results are reported in Figure 14. As n_s increases, which means that more points are sampled to calculate the approximate probablity, obviously more execution time is needed. On one hand, the average Pre gets a little smaller since a little more compressed trajectories, whose corresponding raw trajectories are not overlapped with the query region, are determined to be in the range query result set. On the other hand, the average Rec gets a little larger because a little more compressed trajectories, whose corresponding raw trajectories are overlapped with the query region, can be found by our RQC algorithm. And the average F_1, a comprehensive evaluation metric, does not change much. In order to make a good balance between the execution time and the quality of range query result on compressed trajectories, n_s is set to 15 as the default value in other experiments.
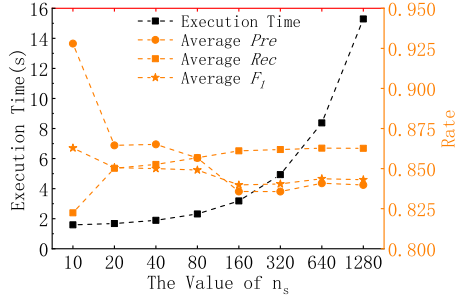
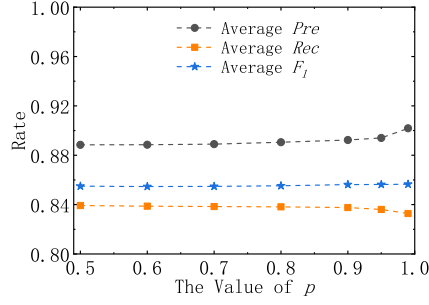**Fig. 14** Evaluation of the execution time, the average Pre, Rec and F_1: varying the value of n_s

Fig. 15 Evaluation of the average Pre, Rec and F_1: varying the value of p

4.3.4 Impacts of the Probability Threshold Value p

For a compressed trajectory, only when the calculated probability of its corresponding raw trajectory overlapped with query region R is larger than p, can this compressed trajectory be in the range query result set. We evaluate the average precision rate Pre, the average recall rate Rec and the average F_1 of query results of 1000 randomly generated range queries on the compressed trajectories whose compression rate is 200 w.r.t. varying p from 0.5 to 0.99, and the results are shown in Figure 15. With the increase of p, we can see that for the average precision rate Pre and the average recall rate Rec, especially the average F_1, they do not change much, which shows that changing the value of p has quite limited influence on the quality of range query results on compressed trajectories.

4.3.5 Impacts of ASP_tree

How much ASP_tree index can accelerate the range query processing is studied first. The execution time of 1000 randomly generated range queries is evaluated w.r.t. varying the compression rate, and the results are reported in Figure 16. By using ASP_tree, the range query processing can be accelerated extremely. At least 99.98% of the execution time can be easily saved, and the acceleration gets more obvious with the decrease of the compression rate of the queried trajectories.

For each node in ASP_tree, if there are more than ξ endpoints of all line segments falling in the corresponding region of this node, then this node is a non-leaf node with 4 child nodes. So the threshold value ξ controls the average height of ASP_tree. The average height of ASP_tree is defined as the average height of all leaf nodes, and the height of the root node is defined as 1. We evaluate the average height of ASP_tree and the execution time of 1000 randomly generated range queries on the compressed trajectories whose compression rate is 200 w.r.t. varying ξ, and the results are reported in Figure 17. The heights of all leaf nodes are always the same, which means that ASP_tree
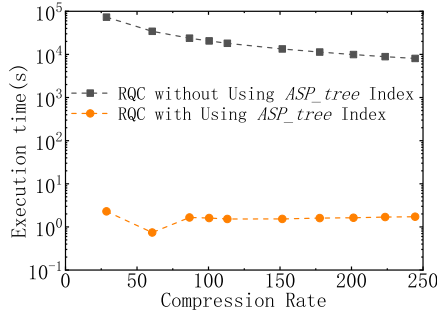
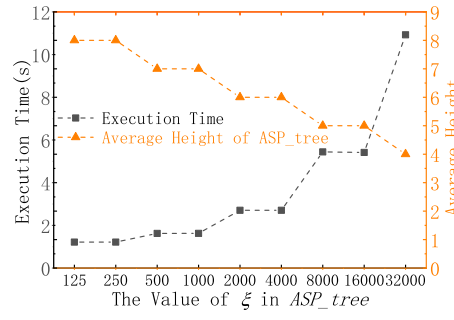**Fig. 16** Efficiency evaluation: *ASP_tree* index when varying the compression rate

Fig. 17 Evaluation of the execution time and the average height of *ASP_tree*: varying the value of ξ in *ASP_tree*

is always well balanced, our region partitioning strategy dose work, and data skew is perfectly avoided. We can see that ξ controls the average height of *ASP_tree*, and the average height declines with the increase of ξ. The average height has direct impact on the execution time of range queries. Less execution time is needed by these range queries with the increase of the average height of *ASP_tree*, because much more compressed trajectories not in the final result set are directly filtered out by searching *ASP_tree*.

5 Related Work

Trajectory Compression algorithms in online mode. Able to be applied in much more application scenarios, some trajectory compression algorithms in online mode based on different accuracy loss metrics have been proposed and attract people's attention. PED, SED, DAD and LISSED are 4 frequently-used accuracy loss metrics, which measure the degree of the accuracy loss after a trajectory is compressed. There is no clear evidence that there exists an accuracy loss metric superior to all the others in the literature. On one hand, DAD, a direction-based distance, is defined based on the greatest angular difference between two directions. Since DAD does not provide any error guarantee on the distance, for the compressed trajectories generated by compression algorithms based on DAD[5, 15, 16, 22], the main weakness is that a discarded point may be too far away from its corresponding line segment. So such a discarded point can not be approximately represented by its corresponding line segment well. On the other hand, PED, SED, LISSED and PSED are all position-based distances, which are defined based on the Euclidean distance between each discarded point and its "mapped" position on the compressed trajectory. But they do not provide any error guarantee on the direction information[5]. For SED[23–26] and LISSED[3, 4], the time attribute of each discarded point is used to find its synchronized point on the corresponding line segment. Introduced in Section 2.2, PED[9, 13, 17–20, 23] is adopted by most existing line simplification methods[18], and there are mainly 4 most popular trajectory

compression algorithms in online mode using PED as their error metric, i.e. OPW[17, 23], BQS[19, 20], FBQS[19, 20] and OPERB[18]. Among them, OPW is proposed the most early and it compresses a trajectory segment as long as possible into an ϵ-error-bounded line segment, with ϵ being the upper bound of PED value. During each loop iteration to compress a trajectory segment into a line segment, BQS builds a virtual coordinate system centered at the starting point at the beginning. In each of four quadrants, BQS establishes a rectangular bounding box and two bounding lines so that in some cases, a point can be quickly decided for removal or preservation without needing expensive error calculation. For FBQS, a fast version of BQS, a raw trajectory point is directly reserved when error calculation is needed in BQS. So error calculation is no longer needed in FBQS. So, FBQS is a little faster than BQS at the expense of the larger size of the generated compressed trajectories. OPERB is based on a novel distance checking method and a directed line segment is used to approximate the buffered points. For BQS and FBQS, the accuracy loss of the compressed trajectories is relatively small, but their time costs are both extremely high. For OPERB and OPW, relatively less execution time is needed, but at the expense of the extremely high accuracy loss of the compressed trajectories. So ROCE, which makes the best balance among the accuracy loss, the time cost and the compression rate, is badly needed.

Query on trajectories. Large trajectory data facilitates various real-world applications, such as route planing, trajectory pattern mining and travel time prediction, and lots of attention has been drawn in queries on trajectories, such as [1, 10, 27, 30, 31]. For example, by analyzing large amounts of historical trajectories, Dai et al. [7] and Dai et al. [6] study how to provide better navigation services for a driver on considering time cost, fuel consumption and the preference of the driver. In trajectories generated by the same person, there must be some common features hidden. And Wu et al. [29] and Jin et al. [14] investigate the potential for historical trajectories accumulated from different sources to be linked so as to reconstruct a larger trajectory of a single person. Detecting anomalous trajectories (i.e. detours) has become an important and fundamental concern in many real-world applications. Liu et al. [21] proposes a novel deep generative model to solve the problem of online anomalous trajectory detection. Real-time co-movement pattern mining for trajectories is to discover co-moving objects that satisfy specific spatio-temporal constraints in real time, and it serves a range of real-world applications, such as traffic monitoring and management. Targeting the visualization and interaction with such co-movement detection on streaming trajectories, Fang et al. [11] proposes a real-time co-movement pattern mining system to handle streaming trajectories. As more and more trajectories are being generated, the amount of trajectories to be queried usually exceeds the storage and processing capability of a single machine. But the situations, where the queried and analyzed trajectories are too much to be queried or they are all compressed trajectories, are considered in none of these works. To address these, Shang et al. [28] adopts a different strategy from us, and proposes a distributed in-memory trajectory analytics system to support large-scale trajectory analytics in distributed

environments. This strategy and our queries on compressed trajectories are orthogonal to each other, and may be combined with each other to further improve the efficiency of queris on trajectory.

6 Conclusions

For existing trajectory compression algorithms in online mode, either too much execution time is needed, or the accuracy loss of the compressed trajectories is not tolerable. To address this, this paper has presented ROCE, an efficient compression algorithm. A range query serves as a primitive, yet essential operation on trajectories. Using the traditional criteria on compressed trajectories will lead to that many trajectories will be missing in the result set. To solve this problem, we propose a new criteria based on the probability and an efficient range query processing algorithm RQC. An efficient index *ASP_tree* and lots of novel techniques are also presented to accelerate the processing of trajectory compression and range queries. Extensive experiments have been conducted using real-life trajectory datasets. The results demonstrate that ROCE makes the best balance among the accuracy loss, the time cost and the compression rate, and the difference between range query results on compressed trajectories and those on the corresponding raw trajectories is reduced greatly by using our RQC algorithm.

In the future, we consider to propose an optimal compression algorithm based on PSED, which can compress a trajectory into an ϵ-error-bounded compressed trajectory with the smallest number of consecutive line segments. And this compression algorithm should also be of high-efficiency. On compressed trajectories, more kinds of queries should be also well studied to reduce the difference between the query results on compressed trajectories and those on the corresponding raw trajectories.

References

1. Ali ME, Eusuf SS, Abdullah K, Choudhury FM, Culpepper JS, Sellis T (2019) The maximum trajectory coverage query in spatial databases. Proceedings of the VLDB Endowment 12(3)
2. Brščić D, Kanda T, Ikeda T, Miyashita T (2013) Person tracking in large public spaces using 3-d range sensors. IEEE Transactions on Human-Machine Systems 43(6):522–534
3. Cao W, Li Y (2017) Dots: An online and near-optimal trajectory simplification algorithm. Journal of Systems and Software 126:34–44
4. Chen M, Xu M, Franti P (2012) A fast $o(n)$ multiresolution polygonal approximation algorithm for gps trajectory simplification. IEEE Transactions on Image Processing 21(5):2770–2785, DOI 10.1109/TIP.2012.2186146
5. Cheng L, Wong RCW, Jagadish H (2013) Direction-preserving trajectory simplification. Proceedings of the VLDB Endowment 6(10):949–960

6. Dai J, Yang B, Guo C, Ding Z (2015) Personalized route recommendation using big trajectory data. In: 2015 IEEE 31st international conference on data engineering, IEEE, pp 543–554
7. Dai J, Yang B, Guo C, Jensen CS, Hu J (2016) Path cost distribution estimation using trajectory data. Proceedings of the VLDB Endowment 10(3):85–96
8. Dong K, Zhang B, Shen Y, Zhu Y, Yu J (2018) Gat: A unified gpu-accelerated framework for processing batch trajectory queries. IEEE Transactions on Knowledge and Data Engineering 32(1):92–107
9. Douglas DH, Peucker TK (1973) Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica: the international journal for geographic information and geovisualization 10(2):112–122
10. Duan L, Pang T, Nummenmaa J, Zuo J, Zhang P, Tang C (2018) Busolap: A data management model for non-on-time events query over bus journey data. Data Science and Engineering 3(1):52–67
11. Fang Z, Gao Y, Pan L, Chen L, Miao X, Jensen CS (2020) Coming: A real-time co-movement mining system for streaming trajectories. In: Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data, pp 2777–2780
12. Flack A, Fiedler W, Blas J, Pokrovski I, Mitropolsky B, Kaatz M, Aghababyan K, Khachatryan A, Fakriadis I, Makrigianni E, Jerzak L, Shamin M, Shamina C, Azafzaf H, Feltrup-Azafzaf C, Mokotjomela T, Wikelski M (2015) Data from: Costs of migratory decisions: a comparison across eight white stork populations
13. Hershberger JE, Snoeyink J (1992) Speeding up the Douglas-Peucker line-simplification algorithm. University of British Columbia, Department of Computer Science Vancouver, BC
14. Jin F, Hua W, Xu J, Zhou X (2019) Moving object linking based on historical trace. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE), IEEE, pp 1058–1069
15. Ke B, Shao J, Zhang Y, Zhang D, Yang Y (2016) An online approach for direction-based trajectory compression with error bound guarantee. In: Asia-Pacific Web Conference, Springer, pp 79–91
16. Ke B, Shao J, Zhang D (2017) An efficient online approach for direction-preserving trajectory simplification with interval bounds. In: 18th IEEE MDM, pp 50–55
17. Keogh E, Chu S, Hart D, Pazzani M (2001) An online algorithm for segmenting time series. In: Proceedings ICDM, pp 289–296
18. Lin X, Ma S, Zhang H, Wo T, Huai J (2017) One-pass error bounded trajectory simplification. Proceedings of the VLDB Endowment 10(7):841–852
19. Liu J, Zhao K, Sommer P, Shang S, Kusy B, Jurdak R (2015) Bounded quadrant system: Error-bounded trajectory compression on the go. In: IEEE 31st ICDE, pp 987–998

20. Liu J, Zhao K, Sommer P, Shang S, Kusy B, Lee JG, Jurdak R (2016) A novel framework for online amnesic trajectory compression in resource-constrained environments. IEEE Transactions on Knowledge and Data Engineering 28(11):2827–2841
21. Liu Y, Zhao K, Cong G, Bao Z (2020) Online anomalous trajectory detection with deep generative sequence modeling. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE), IEEE, pp 949–960
22. Long C, Wong CW, Jagadish HV (2014) Trajectory simplification: On minimizing the directionbased error. Proceedings of the VLDB Endowment 8(1):49–60
23. Meratnia N, Rolf A (2004) Spatiotemporal compression techniques for moving point objects. In: International Conference on Extending Database Technology, Springer, pp 765–782
24. Muckell J, Hwang JH, Patil V, Lawson CT, Ping F, Ravi S (2011) Squish: an online approach for gps trajectory compression. In: Proceedings of the 2nd international conference on computing for geospatial research & applications, pp 1–8
25. Muckell J, Olsen PW, Hwang JH, Lawson CT, Ravi S (2014) Compression of trajectory data: a comprehensive evaluation and new approach. GeoInformatica 18(3):435–460
26. Potamias M, Patroumpas K, Sellis T (2006) Sampling trajectory streams with spatiotemporal criteria. In: 18th International Conference on Scientific and Statistical Database Management (SSDBM'06), IEEE, pp 275–284
27. Shang S, Chen L, Wei Z, Jensen CS, Zheng K, Kalnis P (2017) Trajectory similarity join in spatial networks. Proceedings of the VLDB Endowment 10(11)
28. Shang Z, Li G, Bao Z (2018) Dita: Distributed in-memory trajectory analytics. In: Proceedings of the 2018 International Conference on Management of Data, pp 725–740
29. Wu H, Xue M, Cao J, Karras P, Ng WS, Koo KK (2016) Fuzzy trajectory linking. In: 2016 IEEE 32nd International Conference on Data Engineering (ICDE), IEEE, pp 859–870
30. Xu J, Bao Z, Lu H (2019) Continuous range queries over multi-attribute trajectories. In: 2019 IEEE 35th International Conference on Data Engineering (ICDE), IEEE, pp 1610–1613
31. Yuan H, Li G (2019) Distributed in-memory trajectory similarity search and join on road network. In: 2019 IEEE 35th international conference on data engineering (ICDE), IEEE, pp 1262–1273
32. Zhang B, Shen Y, Zhu Y, Yu J (2018) A gpu-accelerated framework for processing trajectory queries. In: IEEE 34th ICDE, pp 1037–1048
33. Zhang D, Ding M, Yang D, Liu Y, Fan J, Shen HT (2018) Trajectory simplification: an experimental study and quality analysis. Proceedings of the VLDB Endowment 11(9):934–946