This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2914449, IEEE Transactions on Knowledge and Data Engineering

Reference-based Framework for Spatio-temporal Trajectory Compression and Query Processing

Kai Zheng, Yan Zhao*, Defu Lian, Bolong Zheng, Guanfeng Liu, and Xiaofang Zhou *Fellow, IEEE*

Abstract—The pervasiveness of GPS-enabled devices and wireless communication technologies results in massive trajectory data, incurring expensive cost for storage, transmission, and query processing. To relieve this problem, in this paper we propose a novel framework for compressing trajectory data, REST (Reference-based Spatio-temporal trajectory compression), by which a raw trajectory is represented by concatenation of a series of historical (sub-)trajectories (called reference trajectories) that form the compressed trajectory within a given spatio-temporal deviation threshold. In order to construct a reference trajectory set that can most benefit the subsequent compression, we propose three kinds of techniques to select reference trajectories wisely from a large dataset such that the resulting reference set is more compact yet covering most footprints of trajectories in the area of interest. To address the computational issue caused by the large number of combinations of reference trajectories that may exist for resembling a given trajectory, we propose efficient greedy algorithms that run in the blink of an eye and dynamic programming algorithms that can achieve the optimal compression ratio. Compared to existing work on trajectory compression, our framework has few assumptions about data such as moving within a road network or moving with constant direction and speed, and better compression performance with fairly small spatio-temporal loss. In addition, by indexing the reference trajectories directly with an in-memory R-tree and building connections to the raw trajectories with inverted index, we develop an extremely efficient algorithm that can answer spatio-temporal range queries over trajectories in their compressed form. Extensive experiments on a real taxi trajectory dataset demonstrate the superiority of our framework over existing representative approaches in terms of both compression ratio and efficiency.

Index Terms—Reference Trajectory, Spatio-temporal Trajectory, Compression.

1 Introduction

The prevalent use of various mobile devices, such as smartphones, on-board diagnostics, personal navigation devices, and wearable smart devices, has resulted in massive amount of trajectory data. While they contain a wealth of mobility information and offer great opportunities for heightening our understanding about human mobilities, transmitting and storing raw trajectory data consumes too much network bandwidth and storage capacity [22]. This is calling for effective trajectory compression techniques that can remove redundant and inessential samples from raw trajectory data to reduce the storage cost while preserving the utility of data.

- *K. Zheng and Y. Zhao made equal contribution.*
- *K. Zheng is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu, China and Zhejiang Lab, China. Email: zhengkai@uestc.edu.cn.*
- *Y. Zhao is with the School of Computer Science and Technology, Soochow University, Suzhou, China and Zhejiang Lab, China. Email: zhaoyan@suda.edu.cn. * Y. Zhao is the corresponding author of the paper.*
- *D. Lian is with the School of Computer Science and Technology and School of Data Science, University of Science and Technology of China, Hefei, China. Email: dove.ustc@gmail.com.*
- *B. Zheng is with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, China. Email: bolongzheng@hust.edu.cn.*
- *G. Liu is with the Department of Computing, Macquarie University, Sydney, NSW Australia. Email: guanfeng.liu@mq.edu.au.*
- *X. Zhou is with the University of Queensland, Brisbane, Australia and Zhejiang Lab, China. Email: zxf@itee.uq.edu.au.*

Trajectory data compression approaches can be generally divided into two categories: spatial and spatio-temporal compression. Treating trajectories as polylines, spatial compression methods are also known as line simplification algorithms (e.g., Douglas-Peucker (DP) [7] and Bellman's algorithm [2]), which discard some samples within a given spatial deviation threshold from its original locations. However, trajectories are spatio-temporal records of moving objects, in which temporal information is also critical in many applications such as trajectory monitoring [15] and location tracking [20]. Ignoring temporal information during compression may produce unbounded erroneous results when querying the decompressed data. Therefore, recent studies focus on spatio-temporal compression algorithms [12], [22], [24], [27], [30], which adopt spatio-temporal criteria to bound the compression loss. The common feature of the above approaches is that they just exploit the spatio-temporal characteristics of the single trajectory to be compressed and assume moving objects do not change speed and/or direction frequently while traveling. However, this is quite an optimistic assumption for objects moving with complicated traffic condition, which is why those algorithms cannot achieve high compression ratio on real-world trajectory data. Recently, Song et al [29] propose a data-driven approach, called PRESS, that leverages shortest path and frequent trajectory pattern to compress trajectories in a road network. Nevertheless, there are two key prerequisites for PRESS to work properly. First, object movements must be constrained by a network, which does not apply for a wide

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2914449, IEEE Transactions on Knowledge and Data Engineering

range of free-space moving objects such as animals, flying objects, handwriting trajectories and so on. Second, the network should be relatively stable so that the compressed trajectories can be used properly. However, it is not uncommon the topology of road network changes frequently in developing areas (e.g., major cities of China)[1]. Since PRESS relies on precomputed all-pair shortest paths, it has to recompute a large number of shortest paths every time the network updates, which is a time consuming process. Moreover, it takes tremendous space to store the all-pair shortest paths for each version of road network[2].

In this paper, we propose a completely data-driven framework, called REST (Reference-based Spatio-temporal trajectory compression), for trajectory compression. There are a few advantages for REST compared to existing work. First, trajectories can originate from any kind of space (either constrained or non-constrained space). Second, it includes both spatial-only and spatio-temporal compression algorithms. Third, it only takes small amount of memory to store the auxiliary information (i.e., reference set). Fourth, the trajectory data are indexable and usable in their compressed form. This framework is based on some prior studies that find human mobilities have inherent high-level spatial and temporal regularity [8] (i.e., people have high probability to repeat similar travel patterns) and highly skewed travel distribution [35] (i.e., different people often take similar routes when traveling between certain locations). Therefore it is feasible to extract a relatively small collection of trajectories, named reference trajectories, which "covers" most trajectories in the region of interest. Then given a new trajectory in the same area, there is high chance we can use a proper concatenation of reference (sub-)trajectories to resemble, at least partially, the given one. Compared with the raw format that keeps every location sample, this representation saves significant space since only a series of identifiers and offsets of the reference (sub-)trajectories need to be recorded.

As shown in Figure 1, the REST framework is comprised of two components: reference set construction and reference-based compression. The first component aims to build a reference system, where the challenge is how to trade off high coverage and low redundancy in the reference set such that subsequent compression can be performed more effectively and efficiently. To this end, we present three kinds of approaches including Frequent Pattern-based Approach, Redundancy Reduction Approach and Compression-based Approach, which use different strategies to select a compact yet expressive reference set from a much larger but more redundant training dataset. The the second component needs to tackle the computational issue in the great number of reference trajectories we can use to represent a given trajectory. For the sake of efficiency, we propose greedy algorithms that try to represent the longest possible sequence of samples with a single reference trajectory. We also develop optimal algorithms to calculate the minimal storage cost of the compressed trajectory and obtain the corresponding optimal combination of reference trajectories.

1. TomTom claims their digital maps have fixes and updates every week. https://www.tomtom.com/en_au/mydrive-connect/

2. Keeping all-pair shortest path requires $O(|V|^2)$ space where V represents the vertex set in a road network.

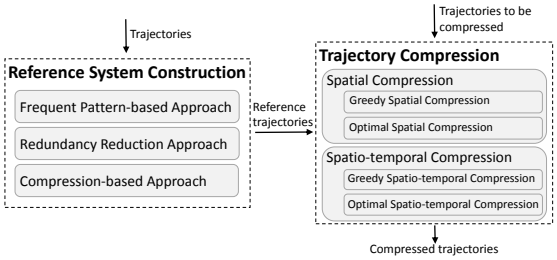


Fig. 1. REST Framework Overview

Though our preliminary work [32] has already optimally compressed the spatio-temporal information of trajectory data with reference trajectories, it fails to demonstrate the key utility of compressed trajectory data. Trajectory utility mainly depends on the effective and efficient trajectory query processing in trajectory databases, which aims to evaluate the spatio-temporal relationships among spatial data objects. However, it is a challenging task to answer the trajectory queries efficiently due to the inherent difficulties in indexing trajectories as well as the new complexity introduced by the compressed trajectories, which are in the form of sequence of reference trajectories. To tackle this problem, we develop an effective hybrid index structure to support efficient query processing over compressed trajectories without fully decompressing the data and then present the query processing based on the index structure in Section 6.

As a summary, the major value-added extension over our preliminary work [32] in REST framework are four folds.

1) We demonstrate REST can support the classical spatio-temporal queries (e.g., range queries) in trajectory databases without fully recovering the compressed trajectories.

2) We design an effective hybrid index structure, consisting of both R-tree and inverted index, to support efficient query processing over the compressed trajectories.

3) We give more justifications about the custom defined error metrics and discussions on the applicability of our REST framework to other trajectory similarity measures.

4) An extensive experimental study is conducted on two real taxi trajectory datasets to validate the effectiveness and efficiency of the query processing on the compressed trajectories.

The remainder of this paper is organized as follows. Section 2 introduces preliminary concepts, error metric and reference-based compression problem. We then present the construction of reference trajectory set in Section 3. The algorithms for spatial-only compression and spatio-temporal compression are presented in Section 4 and Section 5 respectively, followed by the extension of query processing for the compressed trajectories in Section 6. We report the results from empirical study in Section 7 and survey the related work in Section 8. Section 9 concludes this paper.

2 Problem Statement

In this section, we will present a set of preliminary concepts, introduce the error metric between raw and compressed trajectories, and finally state our problem and goal.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2914449, IEEE Transactions on Knowledge and Data Engineering

2.1 Preliminary

Definition 1 (Raw Trajectory). A raw trajectory of a moving object in 2D Euclidean plane, denoted as T, is a finite sequence of timestamped locations with the form of $((x_1, y_1, t_1), (x_2, y_2, t_2), ..., (x_n, y_n, t_n))$, where (x_i, y_i) stands for the longitude and latitude information of a sampled location at time stamp t_i.

Definition 2 (Sub-Trajectory). A sub-trajectory, denoted by $T^{(i,j)}$, is made of consecutive sample points of T from the i-th to j-th triplet, i.e., $T^{(i,j)} : ((x_i, y_i, t_i), ..., (x_j, y_j, t_j))$.

Trajectory compression is a process to reduce storage cost while keeping utility of a trajectory. Normally there are two measures when comparing trajectory compression methods:

1) *Compression Ratio* measures how much space has been saved by compressing raw trajectories. It is usually defined as the ratio between space costs of raw trajectories and compressed trajectories, i.e., $CR = \frac{space(T)}{space(T')}$.

2) *Compression Loss* measures to what extent a compressed trajectory can be reconstructed to its corresponding raw format. It is usually quantified by a distance between raw trajectory and decompressed trajectory based on some predefined distance function.

These two factors are often trade-off: high compression ratio usually leads to greater compression loss and vice versa. Based on compression loss, trajectory compression can be classified into *lossless compression* and *lossy compression*. Due to the extremely fine granularity (hence almost infinite cardinality) of spatio-temporal dimensions in free space, lossless compression algorithms either are not practical or have extremely low compression ratio. Thus in this paper we resort to *bounded lossy compression* algorithm.

Definition 3 (Bounded Lossy Compression). Given a deviation threshold ϵ, an ϵ-Bounded Lossy Compression algorithm transforms a raw trajectory T into a compressed trajectory T', such that the distance between the reconstructed trajectory T^* and T does not exceed ϵ, i.e., $d(T, T^*) \leq \epsilon$, where d is some predefined distance function for trajectories.

2.2 Error Metric

In this paper, we propose a simple but effective variant of Dynamic Time Warping (DTW) [3] distance, called MaxDTW, to serve the error metric. It works exactly the same way as DTW in looking for the best alignment between two unsynchronized trajectories, with the only exception that it just needs to record the maximum distance among all matched pairs instead of the sum. More formally,

Definition 4 (MaxDTW). Given two trajectories $T_a = (p_1, p_2, ..., p_n)$ and $T_b = (q_1, q_2, ..., q_m)$, the MaxDTW distance between them is defined as follows:

$$MaxDTW(T_a, T_b) = \begin{cases} 0, \text{if } T_a = T_b = \emptyset \\ +\infty, \text{if } T_a = \emptyset \text{ or } T_b = \emptyset \\ max\{d(p_n, q_m), Q(p_n, q_m)\}, \quad \text{otherwise} \end{cases}$$

$$Q(p_n, q_m) = min \begin{cases} MaxDTW(T_a^{(1,n-1)}, T_b^{(1,m-1)}) \\ MaxDTW(T_a^{(1,n-1)}, T_b^{(1,m)}) \\ MaxDTW(T_a^{(1,n)}, T_b^{(1,m-1)}) \end{cases}$$

where $d(a, b)$ is a given distance between point a and b.

Similar to DTW, we can use a dynamic programming algorithm [3] to compute MaxDTW.

While our framework can be applied to both conventional DTW and the newly defined MaxDTW (will be discussed in Section 4.1), we chose to define a custom metric MaxDTW because this is easier for user to set the error threshold. To see the reason, DTW is calculated based on distance aggregation (sum of distance), so it is dependent on not only the closeness between two trajectories but also their size (i.e., number of sample points). Therefore, when a user wants to compress a set of trajectories, she needs to set a different error threshold for each individual trajectory as they are of different sizes. For instance, we should set a lower error threshold for a trajectory with 10 points and higher error threshold for another trajectory with 100 points if we expect them to be compressed with similar quality. Of course, this problem can be solved by using the average DTW, i.e., DTW/(size of trajectory), which is essentially the same as our proposed MaxDTW. The rational behind MaxDTW is the furthest pair-wise distance between two trajectories in their best DTW alignment, which is independent of the size. This also gives the user an intuitive view of the closeness they should expect to see between the decompressed trajectory and its corresponding raw trajectory.

2.3 Reference-based Compression

As observed in previous studies [35], there is strong bias when most drivers plan their routes, which means given a new trajectory it is very likely to find from a historical trajectory dataset a few trajectories that resemble, at least partially, the given one. We name these trajectory set as *reference trajectory set*, denoted by R, which can be generated from a historical trajectory dataset in the region of interest (e.g., where the trajectories to be compressed also reside). While it is difficult to define the optimality of R, there are two qualitative measures for a good one – high coverage and low redundancy. Here *high coverage* means it has enough power to represent a given trajectory in the same area of interest, which heavily affects the compression ratio. *Low redundancy* means most trajectories in R are quite unique in terms of their geographical locations since overlapping reference trajectories do not increase the expressive power and make the compression inefficient. In the rest of the paper we use *reference trajectory set* and *reference set* interchangeably when no ambiguity is caused.

Problem Statement: Given a reference trajectory set R, a trajectory T to be compressed and an error threshold ϵ, a reference-based compression algorithm uses a selected subset of R, or their sub-trajectories whenever possible, to represent T, denoted as T', and guarantees that the distance between T' and T does not exceed ϵ.

3 Reference Set Construction

In this section, we propose three methods to build a compact and expressive reference set, which is relatively stable and do not require frequent updates to compress new data. Figure 2 shows a training trajectory set, in which each trajectory sample is labeled by $T_i^{(j)}$ indicating the j-th sample of trajectory T_i. Besides, the first sample of each trajectory is

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2914449, IEEE Transactions on Knowledge and Data Engineering

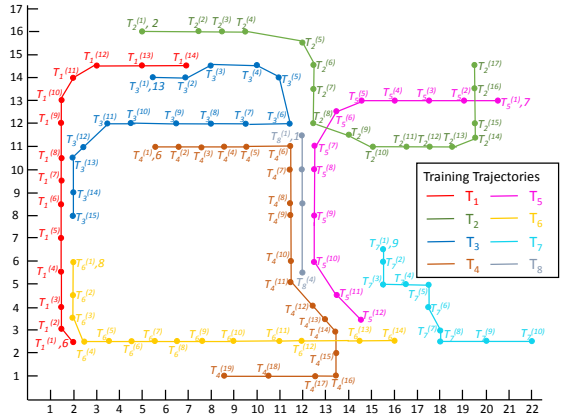


Fig. 2. Running example

companied by a number that indicates the start time stamp. We assume the sampling interval is 1 time unit. For instance, $T_2^{(2)}$ is the second sample of T_2 whose time stamp is 3.

3.1 Frequent Pattern-based Approach (FPA)

Previous studies have shown that trajectories of moving objects often follow certain patterns, such as commuter patterns, peak/off peak patterns, weekend patterns, etc. Therefore a natural thought is to leverage these frequent patterns for building a reference set. Given a trajectory dataset, its frequent pattern is a set of sub-trajectories of which the occurrence frequencies exceed a certain support threshold. Inspired by [18], we first introduce *calculating point* and *calculating trajectory* to discretize trajectories into sequential data and then apply Sequential Pattern Mining algorithms [1] to find frequent sub-trajectories. Specifically, given a sample point set $P = \{p_1, p_2, ..., p_n\}$, p_i is called a calculating point a if $|p_j.x - p_i.x| \leq \epsilon_s$ and $|p_j.y - p_i.y| \leq \epsilon_s$. Then all p_j can be represented by a. A calculating trajectory is a sequence formed by the calculating points chronologically. The following steps are performed to find all of the frequent sub-trajectories:

1) Find all calculating points and calculating trajectories.

2) Given the minimum support threshold, the frequent calculating points are obtained by scanning all calculating points.

3) Remove non-frequent calculating points, and obtain frequent calculating sub-trajectories.

4) Remove the frequent calculating sub-trajectories who is sub-trajectories of another frequent calculating trajectories.

5) Return all the frequent sub-trajectories.

3.2 Redundancy Reduction Approach

Since only using frequent patterns may result in low coverage, we next present two variant methods to achieve higher coverage and reduce redundancy (to some extent) simultaneously.

3.2.1 Segment Redundancy Reduction (SRR)

Given a training trajectory set, we aim to extract a set of non-redundant sub-trajectories. First we define redundant segment in below,

Definition 5 (Redundant Segment). Given a minimum length threshold η and a distance threshold ϵ_s, two sub-trajectories (segments) with the same number of samples, i.e., $T_a^{(i,i+m)}$ and $T_b^{(j,j+m)}$, are said to overlap with each other if $m \geq \eta$ and their maximum pairwise distance $d_{max} = \max_{0 \leq k \leq m} d(T_a.p_{i+k}, T_b.p_{j+k}) \leq \epsilon_s$. If a sub-trajectory s overlaps with any existing sub-trajectory in a reference set R, s is called a redundant segment.

η is to avoid the existence of too many short segments. The basic idea of our approach is to eliminate all the redundant segments of training trajectories and use remaining segments as the reference set.

3.2.2 Trajectory Redundancy Reduction (TRR)

The reference set constructed by SRR algorithm may end up with too many short segments if η is too small, or too many whole trajectories otherwise (since it gets harder to identify long segment overlap). We present an alternative approach to reduce redundancy by treating each trajectory as atomic, i.e., either use the entire one or nothing. The redundant trajectory is defined as follows:

Definition 6 (Redundant Trajectory). Given an overlap threshold θ, a distance threshold ϵ_s, a trajectory T is called redundant if the overlap portion between T and R, denoted as $L(T, R)$, exceeds θ, where $L(T, R)$ is calculated as the portion of samples in T that are sufficiently close to any samples in R, i.e.,

$$L(T, R) = \frac{|p \in T | \exists q \in R, d(p, q) \leq \epsilon_s|}{|T|} > \theta \quad (1)$$

The TRR algorithm checks every training trajectory T and calculates the overlap portion with R. If the value is below θ, T is added into R as a reference trajectory.

3.3 Compression-based Approach (CA)

As the ultimate goal of building a reference set is to achieve high compression ratio, we can compress a training trajectory against the current reference set with the spatial compression algorithm proposed in the following section and record the compression ratio. If the ratio is high enough, that means the training trajectory can be well described by existing reference trajectories, i.e., it is redundant; otherwise, it is non-redundant. The non-redundant training trajectory will be added into the current reference set.

4 Spatial Compression

In this section, ignoring the time information, we compress a given trajectory using as few reference trajectories as possible to minimize the space cost.

4.1 Matchable Reference Trajectory

We will firstly introduce *matchable reference trajectory* – a basic concept that will be used throughout our algorithms.

Definition 7 (Matchable Reference Trajectory (MRT)). Given a sub-trajectory $T^{(i,j)}$ and a spatial deviation threshold ϵ_s, its matchable reference trajectory set, denoted as $M(T^{(i,j)})$, includes all the reference sub-trajectories with less-than-ϵ_s MaxDTW distance with $T^{(i,j)}$, i.e.,

$$M(T^{(i,j)}) = \Big\{ \mathbb{T}^{(k,g)} \Big| \mathbb{T} \in R, 1 \leq k \leq g \leq |\mathbb{T}|, MaxDTW(T^{(i,j)}, \mathbb{T}^{(k,g)}) \leq \epsilon_s \Big\} \quad (2)$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2914449, IEEE Transactions on Knowledge and Data Engineering

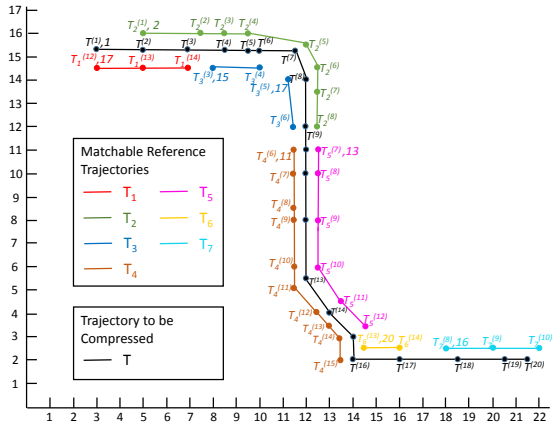


Fig. 3. Matchable Reference Trajectories

Here each MRT $\mathbb{T}^{(k,g)}$ is recorded as a triplet $(\mathbb{T}.id, k, g)$ (costing 8 bytes). To retrieve the MRTs, we propose an efficient method based on the following observation.

Lemma 1. Any sub-trajectory of the MRT of $T^{(i,j)}$ is also an MRT of sub-trajectory of $T^{(i,j)}$.

The proof is directly from the property that MaxDTW distance between longer sequences upper bounds their subsequences. Therefore the MRT set of $T^{(i,j)}$ can be more easily derived by joining the MRT sets of its sub-trajectories. Algorithm 1 is proposed based on this intuition. First, the MRT sets of all the length-2 sub-trajectories is obtained and added to a hash set $M(T^{(i,j)})$ (lines 1-2). Then for each length-n ($2 < n \leq |T|$) sub-trajectory $T^{(i,j)}$, we check if existing MRT of $T^{(i,j-1)}$ and $T^{(j-1,j)}$ can also be the MRT of $T^{(i,j)}$ (lines 6-9), and verify if a longer MRT can be formed for $T^{(i,j)}$ by joining $M(T^{(i,j-1)})$ and $M(T^{(j-1,j)})$ (lines 10-11). The algorithm can be terminated early if we find the MRT sets of all length-n sub-trajectories are empty since all longer sub-trajectories will not have MRT based on Lemma 1. Given $\epsilon_s = 0.9, \eta = 1$ and the reference set generated by SRR algorithm previously, i.e., $R = \{T_1, T_2, T_3^{(3,15)}, T_4, T_5, T_6, T_7\}$, Figure 3 illustrates the MRTs for T, to name a few, $M(T^{(2,9)}) = \{T_2^{(1,8)}\}$, $M(T^{(10,15)}) = \{T_4^{(6,14)}, T_5^{(7,12)}\}$.

Discussion: here we briefly discuss the applicability of our framework to other trajectory similarity measure. Since the correctness of our following algorithms rely on Lemma 1, which states the sub-structure optimality in MRT, we only need to examine whether the derived MRT (from Definition 7) based on a given trajectory similarity measure satisfies Lemma 1 or not. Obviously, our framework can be easily adapted to those distance aggregation based trajectory similarity measure such as Euclidean Distance (ED), Edit Distance with Projections (EDwP) [28], Dynamic Time Warping (DTW) [3] and its variants as they all satisfy Lemma 1. However, the generalization to Longest Common Subsequence (LCSS) [31], Edit distance with Real Penalty (ERP) [16] and Edit Distance on Real sequences (EDR) [17] is not straightforward since they are count-based similarity measure, which means a sub-trajectory of MRT may not meet the similarity threshold any more.

4.2 Greedy Spatial Compression

Once the MRT set is obtained, a natural thought is to process the given trajectory in its chronological order and compress the longest possible sub-trajectory with its MRT (selecting an arbitrary MRT if multiple longest MRTs exist), until the last sample point has been reached. This approach is called Greedy Spatial Compression (GSC) algorithm since it seems not a global strategy to combine MRTs in order to achieve the minimal storage cost. However, we will prove later it also yields space optimal compressed trajectory.

Algorithm 1: Matchable Reference Trajectory Search

Input: T, R, ϵ_s
Output: M
1 **for** *each* $T^{(i,i+1)} \in T$ **do**
2 $M(T^{(i,i+1)}) \leftarrow$ MRT set for segment $T^{(i,i+1)}$;
3 **for** $n \leftarrow 3$ *to* $|T|$ **do**
4 **for** *each length-*n *sub-trajectory* $T^{(i,j)} \in T$ **do**
5 **for** $\mathbb{T}_a^{(m,n)}, \mathbb{T}_b^{(s,t)} \in M(T^{(i,j-1)}), M(T^{(j-1,j)})$ **do**
6 **if** $MaxDTW(T^{(i,j)}, \mathbb{T}_a^{(m,n)}) \leq \epsilon_s$ **then**
7 Add $\mathbb{T}_a^{(m,n)}$ into $M(T^{(i,j)})$;
8 **if** $MaxDTW(T^{(i,j)}, \mathbb{T}_b^{(s,t)}) \leq \epsilon_s$ **then**
9 Add $\mathbb{T}_b^{(s,t)}$ into $M(T^{(i,j)})$;
10 **if** $a = b$ *and* $n = s$ **then**
11 Add $\mathbb{T}_a^{(m,t)}$ into $M(T^{(i,j)})$;
12 **if** *no length-*n *sub-trajectory has MRT* **then**
13 Break;
14 **return** M;

Consider the example in Figure 3. With GSC algorithm, we firstly compress $T^{(1,3)}$ with $T_1^{(12,14)}$ since it is the first longest sub-trajectory with non-empty MRT set. Then the remaining part of T are represented by $T_2^{(3,8)}$, $T_4^{(6,15)}$, $T^{(17)}$, $T_7^{(8,10)}$ respectively, resulting in the space cost of 40 bytes, i.e., 25% of its original space (160 bytes).

4.3 Optimal Spatial Compression

In the sequel, we propose a dynamic programming algorithm, called Optimal Spatial Compression (OSC), that aims to minimize the required storage size for T'. Specifically, given a trajectory T and its MRT set $M(T)$, we define $F_T[i]$ as the minimum storage size needed for compressing $T^{(1,i)}$, and $T'^{(1,i)}$ as the corresponding compressed sub-trajectory to achieve this optimum storage. $F_T[i]$ can be derived by the following recursive formula shown in Equation (3).

$$F_T[i] = \begin{cases} 0 & \text{if } i = 0 \\ \min\limits_{1 \leq j \leq i \wedge M(T^{(j,i)}) \neq \emptyset} \{F_T[j-1] + 8\} & \text{otherwise} \end{cases} \quad (3)$$

where the MRT set of single sample point, i.e., $M(T^{(i,i)})$, is manually set to non empty.

It is trivial $F_T[0] = 0$ when $T = \emptyset$, i.e., $i = 0$. When $i > 0$, $F_T[i]$ is computed by picking the minimum of: 1) the optimal storage cost of sub-trajectory $T'^{(1,j-1)}$, i.e., $F_T[j-1]$, plus the storage (8 bytes) for an MRT of sub-trajectory $T^{(j,i)}$ if $M(T^{(j,i)}) \neq \emptyset$ when $1 \leq j < i$; and 2) the optimal storage cost of sub-trajectory $T'^{(1,i-1)}$, i.e., $F_T[i-1]$, plus the storage (8 bytes) of original sample point $p_i \in T$ when $j = i$.

With Equation 3, now we can compute the minimum storage size of compressed trajectory, which is presented in Algorithm 2. Note that we introduce another notation $pre[i]$ for recording the last-to-first sample points having been compressed before achieving $F_T[i]$ to facilitate the reconstruction of the compressed trajectory T' with minimum

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2914449, IEEE Transactions on Knowledge and Data Engineering

Algorithm 2: Optimal Spatial Compression

```
   Input: T, M(T)
   Output: T'
1  T' ← null;
2  F_T[0] ← 0;
3  for i ← 1 to |T| do
4      min ← 8|T|;
5      for j ← 1 to i do
6          if M(T^(j,i)) ≠ ∅ and F_T[j − 1] + 8 < min then
7              min ← F_T[j − 1] + 8;
8              pre[i] ← j − 1;
9      F_T[i] ← min;
10 i ← |T|;
11 while 0 < i ≤ |T| do
12     if pre[i] ← i − 1 then
13         Add p_i into T';
14         i ← i − 1;
15     else
16         Add arbitrary T^(k,g) ∈ M(T^(pre[i]+1,i)) into T';
17         i ← pre[i];
18 return T';
```

storage size. It first initializes $T' = null$ and $F_T[0] = 0$ (lines 1-2). Then the algorithm processes all points of T in the sampling order from 1 to $|T|$, where each recursion starts with initializing the minimal storage size to the whole storage of T, i.e., $8|T|$, and computes $F_T[i]$, $pre[i]$ according to Equation 3 (lines 3-9). Finally, it presents the procedure of construction T' from table pre (lines 10-17). It is easy to analyze the complexity of Algorithm 2 is $O(|T|^2)$. Taking Figure 3 as a case, we can finally get $F_T[20] = 40$ with $T^{(1)}$, $T_2^{(1,8)}$, $T_5^{(7,12)}$, $T_6^{(13,14)}$, $T_7^{(8,10)}$ compressing T based on Equation 3. It is worth noting that there may be more than one combination of reference trajectories and original sample points of T to get the minimum storage size of T'.

Recall the previous example that the compressed trajectory based on GSC algorithm also costs the same space. We show by Lemma 2 that it turns out not to be a coincidence. The proof is omitted due to space limit.

Lemma 2. The compressed trajectory T' generated by GSC is also space optimal.

5 Spatio-temporal Compression

In this section, we extend spatial compression algorithms into spatio-temporal compression algorithms considering both spatial and temporal information.

5.1 Time Correction Cost

Intuitively, given a spatial deviation threshold ϵ_s and a temporal deviation threshold ϵ_t, if the MaxDTW distance between two trajectories T_a and T_b is smaller than ϵ_s, we also need to modify some timestamps of either T_a or T_b such that the maximum time difference between the matching sample pairs (along the optimal alignment path) does not exceed ϵ_t as well. Each correction of time will incur some extra space, denoted by c, to record the position where this correction takes place and the new timestamp. Here we set $c = 4$ bytes to record the index (15 bits) of corrected samples and time (e.g., seconds) of day (17 bits). However, c can be reconfigured to suit different application requirement. The total extra space incurred by rectifying the timestamps of T_b to match T_a is called *Time Correction Cost*, denoted as $C_{T_a}^{\epsilon_t}(T_b)$.

Suppose the matched sample pairs between T_a and T_b when calculating their MaxDTW distance are (a_1, b_1), (a_2, b_2), ..., (a_n, b_n) ($n \leq |T_a + T_b - 1|$). Note that this representation contains replication of original samples, i.e., ..., a_i, ... may refer to the same sample in T_a, since a sample in $T_a(T_b)$ can match multiple samples in $T_b(T_a)$. Then the time correction cost $C_{T_a}^{\epsilon_t}(T_b)$ can be derived by sequentially processing each $(a_i, b_i)(1 \leq i \leq n)$, and performing the following actions:

1) Initialize $t = 0$ to record the most recent correct timestamp of T_b and set $b_0.t = 0$, $C_{T_a}^{\epsilon_t}(T_b) = 0$;

2) If $|t + b_i.t - b_{i-1}.t - a_i.t| \leq \epsilon_t$, update $t = t + b_i.t - b_{i-1}.t$;

3) Otherwise update $t = \max\{a_i.t, b_{i-1}.t\}$, $C_{T_a}^{\epsilon_t}(T_b) = C_{T_a}^{\epsilon_t}(T_b) + c$, and record the actual index of b_i in T_b and the corrected timestamp t.

When applying the above procedure to our spatio-temporal compression, T_a is the given trajectory to be compressed and T_b is a MRT of T_a. Reconstruction of T_a's temporal information with T_b is also straightforward. Scanning from the first point of T_b, we will replace b_i's timestamp with the corrected timestamp if a correction record can be found; otherwise set $b_i^{new}.t = b_{i-1}^{new}.t + b_i^{old}.t - b_{i-1}^{old}.t$.

5.2 Greedy Spatio-temporal Compression

Similar with GSC algorithm, Greedy Spatio-temporal Compression (GSTC) algorithm also iteratively replaces the longest sub-trajectory (i.e., $T^{(i,j)}$) with its MRT. However, instead of choosing an arbitrary MRT for this longest sub-trajectory, GSTC finds the one with least time correction cost in order to minimize the storage for compressed trajectory.

Applying GSTC algorithm on the running example in Figure 3, the selected MRTs and time correction cost are detailed in Table 1. Since each original sample point costs 12 bytes now (with timestamp), the storage cost of T' is 76 bytes, which is 31.67% of the original space cost (240 bytes).

5.3 Optimal Spatio-temporal Compression

In this part we extend OSC algorithm to Optimal Spatio-temporal (OSTC) algorithm in below.

$$F_T[i] = \begin{cases} 0 & \text{if } i = 0 \\ \min\Big\{F_T[i-1] + 12, \min\limits_{1 \leq j < i \wedge M(T^{(j,i)}) \neq \emptyset} \\ \quad \{F_T[j-1] + C_{T^{(j,i)}}(M_{opt}(T^{(j,i)}))\} + 8\Big\} & \text{otherwise} \end{cases}$$

$F_T[i]$ records the minimal space needed for compressing sub-trajectory $T^{(1,i)}$. $F_T[0] = 0$ defines the termination condition. For $i \geq 1$, the algorithm either 1) keeps the original sample $p_i \in T$; or 2) compresses $T^{(j,i)}$ with $M_{opt}(T^{(j,i)})$. In the first case, the final space cost is simply the compressed space of $T^{(1,i-1)}$ plus 12 bytes (for storing p_i). In the second case, the space cost is the minimal sum of compressed space for $T^{(1,j-1)}$, time correction cost of $M_{opt}(T^{(j,i)})$ and space for $M_{opt}(T^{(j,i)})$ (8 bytes). The process of calculating $F_T[|T|]$ and construction of T' is similar to Algorithm 2 and thus omitted here due to space limitation. The time complexity of this algorithm is also $O(|T|^2)$.

Taking the example in Figure 3, we employ OSTC algorithm to calculate the proper MRTs and time correction

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2914449, IEEE Transactions on Knowledge and Data Engineering

TABLE 1
Compressed Trajectory from GSTC

T	$M_{opt}(T^{(i,j)})$	Time Correction	$C_T^{\epsilon_t}$
$T^{(1,3)}$	$T_1^{(12,14)}$	$T_1^{(12)}.t = 1$	4
$T^{(4,9)}$	$T_2^{(3,8)}$	$T_2^{(4)}.t = 6$, $T_2^{(7)}.t = 8$	8
$T^{(10,16)}$	$T_4^{(6,15)}$	$T_4^{(6)}.t = 10$, $T_4^{(9)}.t = 12$, $T_4^{(11)}.t = 13$, $T_4^{(13)}.t = 14$	16
$T^{(17)}$	$T^{(17)}$		0
$T^{(18,20)}$	$T_7^{(8,10)}$	$T_7^{(8)}.t = 18$	4

TABLE 2
Compressed Trajectory from OSTC

T	$M_{opt}(T^{(i,j)})$	Time Correction	$C_T^{\epsilon_t}$
$T^{(1)}$	$T^{(1)}$		0
$T^{(2,9)}$	$T_2^{(1,8)}$	$T_2^{(4)}.t = 6$, $T_2^{(7)}.t = 8$	8
$T^{(10,15)}$	$T_5^{(7,12)}$	$T_5^{(7)}.t = 10$	4
$T^{(16,17)}$	$T_6^{(13,14)}$	$T_6^{(13)}.t = 16$	4
$T^{(18,20)}$	$T_7^{(8,10)}$	$T_7^{(8)}.t = 18$	4

shown in Table 2, and achieve the minimal storage size of T' (i.e., 64 bytes), resulting in an approximately 15.79% reduction compared to GSTC algorithm.

6 Querying Compressed Trajectories

The utility of trajectory compression depends on the space storage reduction as well as the effective and efficient trajectory query processing in trajectory databases. The proposed OSTC algorithm can compress the raw trajectories in such a way that the spatial and temporal information loss can be bounded with a deviation threshold, ϵ (i.e., ϵ_s and ϵ_t), where ϵ can be set as a very small value. This means the spatial paths and timestamps of a raw trajectory can be captured almost exactly when being compressed. Therefore we can not only directly decompress the compressed trajectories for location based service applications, but also use the compressed trajectories for various queries without fully decompressing the data, in which the error of every point in the partially decompressed trajectories is less than ϵ. In this section, we focus on demonstrating that the compressed trajectories can support the range query, a typical trajectory query aiming to evaluate spatio-temporal relationships among spatial data objects. Informally, in Figure 4, given a raw trajectory set (i.e., $\{T_1, T_2, T_3, T_4\}$) and reference trajectory set (i.e., $R = \{\mathbb{T}_1, \mathbb{T}_2, \mathbb{T}_3\}$) generated by SRR algorithm previously, range query searches for raw trajectories that belong to the specified spatio-temporal region.

The query processing is to extract qualitative information from trajectory databases that contain very large numbers of trajectories, whose efficiency depends crucially upon an appropriate index of trajectories. Due to the unique data characteristics of the compressed trajectories (e.g., containing sequence of both reference (sub-)trajectories and original samples of raw trajectories) and the unique query characteristics (e.g., often querying data in an instantaneous/continuous time window), we design a hybrid spatio-temporal index structure by enhancing R-tree [10] with inverted files. The index structure is presented in Section 6.1, followed by the detailed query processing in Section 6.2.

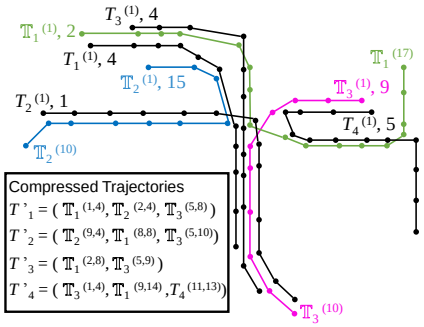


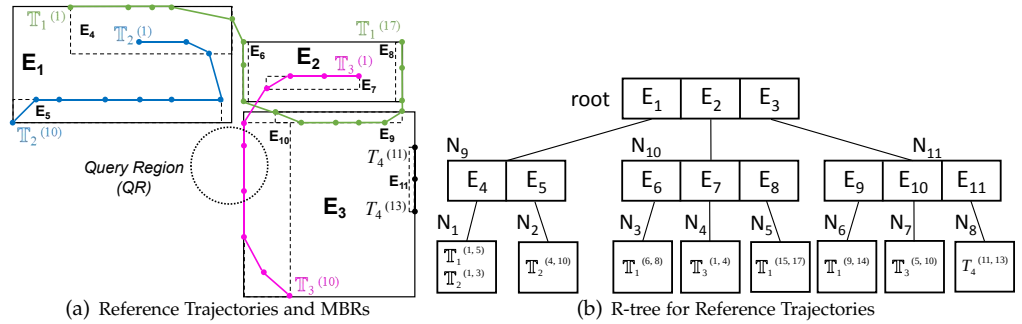
Fig. 4. A Set of Raw Trajectories and Reference Trajectories

6.1 Hybrid Index

In this section, we introduce a hybrid index structure, called IR-tree [19], based on R-tree with each node enhanced with reference to an inverted file for the reference (sub-)trajectories contained in the sub-tree rooted at the node. Compared with the query processing over original trajectories (i.e., process on original data with R-tree), the query processing over compressed trajectories has several non-negligible advantages. First, since the volume of reference trajectories is much less (several orders of magnitude less) than the raw trajectories, there will be less dead space in the IR-tree, which in turn is more efficient for search. Moreover, the R-tree built using reference trajectories is small and compact enough to fully fit into memory to avoid physical I/O operations, so that searching on such an in-memory R-tree is comparatively faster. Our experimental study will also verify these advantages.

In our index structure, a leaf node N contains a number of entries in the form of $(T.id, MBR, ifile)$, in which $T.id$ is an identifier that points to the reference (sub-)trajectories or the uncompressed part of compressed trajectory, MBR is a rectangle with two spatial dimensions, and $ifile$ is a pointer to an inverted file for the reference (sub-)trajectories being indexed. Note that for the uncompressed segments of a compressed trajectory, we regard them as new reference trajectories and organize them in the index structure with the same way as reference trajectories. In this way, a single top-down traversal in this unified IR-tree can cover both compressed and uncompressed parts simultaneously. Taking T_4 in Figure 4 as an example, T_4 can be represented by $(\mathbb{T}_3^{(1,4)}, \mathbb{T}_1^{(9,14)}, T_4^{(11,13)})$, where $T_4^{(11,13)}$ is the uncompressed part of T_4 that is regarded as a new reference trajectory when building the index. The inverted file consists of a hash table and a set of posting lists, wherein the hash table is used to index all the distinct reference (sub-)trajectories in N with a lookup time complexity of $O(1)$. For each reference (sub-)trajectory $\mathbb{T}^{(i,j)}$ in the hash table, we maintain a list of $(T^{(m,n)}, T^{(m)}.t, T^{(n)}.t)$ triples for the raw trajectories relevant with $\mathbb{T}^{(i,j)}$, which are sorted by their start timestamp (i.e., $T^{(m)}.t$). Note that the timestamps of the compressed part (consisting of a set of reference trajectories) in compressed trajectories may not be the actual timestamps. Therefore, before constructing the inverted lists, we recover the timestamps of the compressed part by scanning from the first point of each compressed sub-trajectory T'_{sub}. In particular, the timestamp of $T'^{(i)}_{sub}$ will be replaced with the corrected timestamp when a correction record can be found; otherwise it will be set to $T'^{(i-1)}_{sub}.t + \mathbb{T}^{(i)}.t - \mathbb{T}^{(i-1)}.t$, where

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2914449, IEEE Transactions on Knowledge and Data Engineering



(a) Reference Trajectories and MBRs (b) R-tree for Reference Trajectories

N_1: $\mathbb{T}_1$, $\mathbb{T}_2$: (raw trajectory, start time, end time)	N_2: $\mathbb{T}_2$: (raw trajectory, start time, end time)	N_3: $\mathbb{T}_1$: (raw trajectory, start time, end time)	N_4: $\mathbb{T}_3$: (raw trajectory, start time, end time)	N_5: $\mathbb{T}_1$: (raw trajectory, start time, end time)
$\mathbb{T}_1^{(1,5)}$: ($T_1^{(1,4)}$, 4, 7), ($T_3^{(1,4)}$, 4, 7) $\mathbb{T}_2^{(1,3)}$: ($T_1^{(5,6)}$, 8, 9)	$\mathbb{T}_2^{(4,10)}$: ($T_2^{(1,6)}$, 1, 6), ($T_1^{(7)}$, 10, 10)	$\mathbb{T}_1^{(6,8)}$: ($T_2^{(7)}$, 7, 7), ($T_3^{(5,7)}$, 8, 10)	$\mathbb{T}_3^{(1,4)}$: ($T_4^{(1,4)}$, 5, 8)	$\mathbb{T}_1^{(15,17)}$: NULL

N_6: $\mathbb{T}_1$: (raw trajectory, start time, end time)	N_7: $\mathbb{T}_3$: (raw trajectory, start time, end time)	N_8: T_4: (raw trajectory, start time, end time)	N_9	N_{10}	N_{11}	root
$\mathbb{T}_1^{(9,14)}$: ($T_4^{(5,10)}$,9,14)	$\mathbb{T}_3^{(5,10)}$:($T_2^{(8,13)}$,8,13), ($T_1^{(8,11)}$,11,14), ($T_3^{(8,12)}$, 11,15)	$T_4^{(11,13)}$: ($T_4^{(11,13)}$,15,17)	$\mathbb{T}_1$: N_1 $\mathbb{T}_2$: N_1, N_2	$\mathbb{T}_1$: N_3, N_5 $\mathbb{T}_3$: N_4	$\mathbb{T}_1$: N_6 $\mathbb{T}_3$: N_7	$\mathbb{T}_1$: N_8, N_9, N_{10} $\mathbb{T}_2$: N_8 $\mathbb{T}_3$: N_9, N_{10}

(c) Data Structure of Each Node

Fig. 5. Illustration of Index Structure

$\mathbb{T}$ is the reference trajectory used to compress T'_{sub}.

On the other hand, the entries in a non-leaf node N are of form $(ptr, MBR, ifile)$, in which ptr is the pointer to the child nodes of N, MBR is the minimum bounding rectangle covering all the child nodes, and $ifile$ denotes an inverted file that indexes all the distinct reference (sub-)trajectories within N in a hash table. The IR-tree can be constructed in the similar way of R-tree (i.e., insertion operation) with an exception that the inverted files have to be updated from the leaf to the root when a new reference (sub-)trajectory is added.

Figure 5 presents the illustration of the whole hybrid index structure. Given several raw trajectories, $\{T_1, T_2, T_3, T_4\}$, which can be totally represented by a set of reference trajectories (see Figure 4), we first need to obtain the MBRs of these reference trajectories, as shown in Figure 5(a). Subsequently, the R-tree is constructed based on the distribution of these reference (sub-)trajectories and MBRs in Figure 5(b). The root of the R-tree contains three data entries E_1, E_2 and E_3 referring to the children nodes N_8, N_9 and N_{10} separately. N_8 represents the minimum bounding rectangle of its children nodes N_1 and N_2, and the information of the bounding region is contained in E_1, as with N_9 and N_{10}. The spatial objects (i.e., the segments of the reference trajectories) are referred by the entries of the leaf nodes in R-tree. In Figure 5(c), the invert lists are given for each reference (sub-)trajectories that are used to compress the given raw trajectories.

6.2 Query Processing

After constructing the hybrid index, we next detail the query processing based on the index structure. Since it is expensive to calculate the spatial relationship (e.g., distance, containments, etc) between spatial objects (i.e., a query region and a trajectory in the range query), a query processing algorithm typically adopts a filter-and-refinement approach [6]. In particular, the filter step takes relatively cheap computation cost to find a small set of candidate trajectories that are likely to be the results, which is a super set of the result for the original query. Then these candidate trajectories are further processed using geometric algorithms (e.g., distance calculation) to obtain an actural result at the refinement step.

Subsequently, we discuss how to efficiently process range query based on the IR-tree. The range query aims to retrieve all the raw trajectories within a specified spatio-temporal window, e.g., a sphere of 10km radius around a location (see the circular query region in Figure 5(a)) in the time interval, $\{8, 9, 10\}$. In the filter step, IR-tree is traversed from the root and examines the query region against the MBR in each entry visited to check if they are relevant (i.e., contained/overlapped) with each other. The range query processing finally finds all the relevant leaf nodes. Considering the case in Figure 5(a) and 5(b), N_{10} is visited since its MBR overlaps with the Query Region (QR). Then for the same reason N_7 is visited, and $\mathbb{T}_3^{(5,10)}$ is found from N_7. Finally, we can easily identify the candidate trajectory T_2 in the specific time interval with the inverted lists in Figure 5(c), and examine whether the candidate actually overlaps with the query region in the refinement step.

For other kinds of queries like Trajectory-based query (T-query), we can first simply convert the queries into range query and then apply the filter-and-refinement approach. For instance, given a query trajectory (e.g., T_4 in Figure 4) and a spatial threshold ϵ', a T-query aims to find trajectories that satisfy a given distance function to the query trajectory (e.g., the MaxDTW distance between the result trajectory and the query trajectory is less than ϵ'). Specifically, we

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2914449, IEEE Transactions on Knowledge and Data Engineering

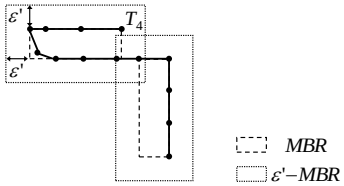


Fig. 6. A Query Trajectory and its ϵ'-MBRs

can first construct several ϵ'-MBRs, each of which extends the original MBR's length and width by $2\epsilon'$ respectively (as shown in Figure 6). These ϵ'-MBRs can be regarded as the query regions, and then we can process range query based on the IR-tree to identify the result trajectories.

7 Experiment

In this section, we conduct extensive experiments to validate the effectiveness of our proposed algorithms. All the algorithms are implemented on an Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHZ with 256 GB RAM.

7.1 Experiment Setup

We use two real trajectory datasets generated by taxis in Beijing and Chengdu respectively over one week. The Beijing dataset, denoted by BJ, contains about 2 million trajectories, and the Chengdu dataset, denoted by CD, contains about 1.4 million trajectories. In both datasets, trajectories in one day are used as training dataset to construct the reference set, and the rest are used to test the performance of different compression algorithms under various parameter settings with the average performance for one day recorded. The ranges and default values (underlined) of all the parameters are summarized in Table 3.

7.2 Experiment Results

7.2.1 Performance of Reference Set Construction

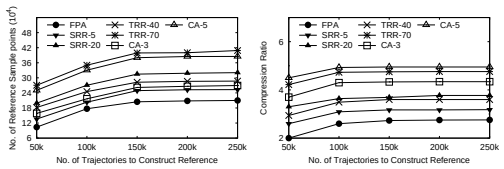
We first evaluate the performance of reference set construction and its impact to spatial compression on BJ dataset. Two metrics, *Size of Reference Set* (the number of sample points in the reference set) and *Compression Ratio (CR)*, are compared in this set of experiments, wherein *CR* measures how much space has been saved by compressing raw trajectories. Specifically, *CR* is defined as the ratio between space costs of raw trajectories and compressed trajectories, i.e., $CR = \frac{space(T)}{space(T')}$, where an original sample point contains its longitude, latitude and timestamp (costing 12 bytes), an MRT $\mathbb{T}^{(k,g)}$ used to represent the raw trajectories is recorded as a triplet $(\mathbb{T}.id, k, g)$ (costing 8 bytes), and a time correction costs 4 bytes. We compare these two metrics among the following methods (specified in Section 3) by varying $|D_R|$, ϵ_s.

1) FPA: FPA with minimum support 100.
2) SRR-5: SRR with minimum sub-trajectory length 5.
3) SRR-20: SRR with minimum sub-trajectory length 20.
4) TRR-40: TRR with overlap threshold 40%.
5) TRR-70: TRR with overlap threshold 70%.
6) CA-3: CA with compression ratio threshold 3.
7) CA-5: CA with compression ratio threshold 5.

Effect of $|D_R|$. In this set of experiment, we study the effect of $|D_R|$. As shown in Figure 7(a), naturally the sizes of reference sets generated from all approaches increase when more training trajectories are used. However, we also notice that the increase becomes slower when $|D_R| > 150k$ since with more reference trajectories accumulated there is increasing chance that subsequently added trajectories are redundant. Among those competing methods, FPA generates the smallest reference set while TRR-70 results the largest followed by CA-5, SRR-20, TRR-40, CA-3 and SRR-5. It is found that the size of reference set almost stops growing beyond 400k, which only takes a few megabytes memory space. From Figure 7(b), the compression effectiveness heavily depends on the size of reference set since a larger reference set normally means greater coverage hence better compression ratio. Compression algorithm performs the worst on the reference set generated by FPA, which aims at capturing the major traveling patterns of training trajectories. Even though CA-5 generates a smaller reference set than TRR-70, it has the best compression ratio, which implies that the reference set generated by CA-5 is of high coverage and low redundancy. This is due to the fact that CA directly optimizes the compression ratio during the construction of reference set while SRR and TRR try to minimize the redundancy.

TABLE 3
Experiment Parameters

Parameters	Values
No. of trajectories to construct reference set $\|D_R\|$	50k, <u>100k</u>, 150k, 200k, 250k
No. of trajectories to be compressed/queried $\|D\|$	<u>200k</u>, 400k, 600k, 800k
Spatial deviation threshold ϵ_s	<u>200m</u>, 400m, 600m, 800m, 1000m
Temporal deviation threshold ϵ_t	30s, <u>60s</u>, 90s, 120s, 150s
Length of trajectory $\|T\|$	50, 100, 150, 200, 250
Size of query window Q_w	2km, <u>4km</u>, 6km, 8km, 10km
Compression ratio CR	2, 4, 6, 8, 10



(a) Size of Reference Set (BJ) (b) Compression Ratio (BJ)

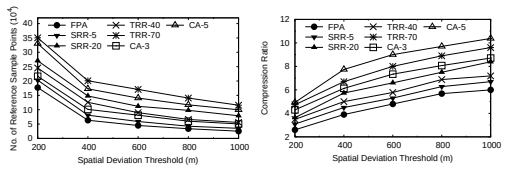
Fig. 7. Performance of Reference Set Construction: Effect of $|D_R|$

Effect of ϵ_s. Next we study the effect of ϵ_s. In Figure 8(a), the sizes of reference sets decrease with the increase of ϵ_s, since ϵ_s affects the granularity of patterns for FPA and the judgment of redundant trajectories for other approaches. Greater ϵ_s means more trajectories become redundant, which in turn results in smaller reference set. On the compression ratio aspect, Figure 8(b) demonstrates that as ϵ_s increases the compression performance improves in spite of smaller reference set since a given trajectory has more chance to match fewer but longer reference trajectories.

7.2.2 Performance of Compression Algorithms

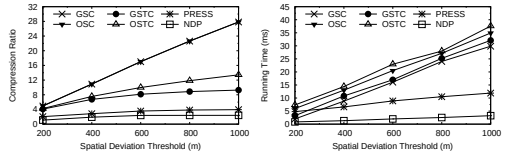
In this part we evaluate the effectiveness (*compression ratio*) and efficiency (*running time*) of the proposed compression algorithms, namely GSC, OSC, GSTC and OSTC, based on the same reference set generated by compress-based approach on both trajectory datasets (i.e., BJ and CD datasets). Moreover, we also implement two representative spatio-temporal compression algorithms, namely Normal Douglas-Peucker (NDP) algorithm based on SED [22] (a spatial trajec-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2914449, IEEE Transactions on Knowledge and Data Engineering

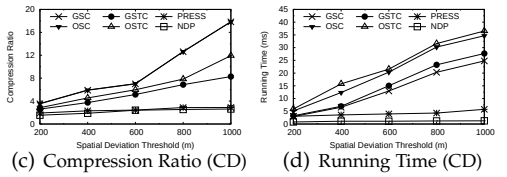


(a) Size of Reference Set (BJ) (b) Compression Ratio (BJ)

Fig. 8. Performance of Reference Set Construction: Effect of ϵ_s



(a) Compression Ratio (BJ) (b) Running Time (BJ)



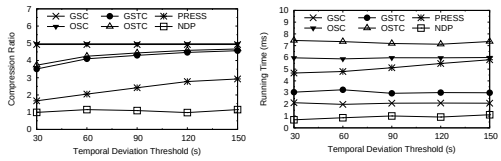
(c) Compression Ratio (CD) (d) Running Time (CD)

Fig. 9. Performance of Compression Algorithms: Effect of ϵ_s

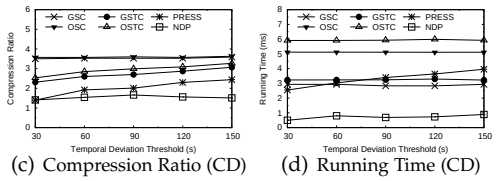
tory simplification algorithm with synchronous Euclidean distance) and PRESS [29] (a spatial-temporal trajectory compression algorithm with the constraint of road network), as our competitors. For NDP, we define ϵ_s as the maximal allowed SED between a raw trajectory and its compressed trajectory. Since the spatial compression component of PRESS is lossless, we use ϵ_s and ϵ_t to represent the error metrics, TSND and NSTD, in the temporal compression component, respectively.

Effect of ϵ_s. As expected, compression ratio of all algorithms gradually increases as ϵ_s grows (see Figure 9(a) and 9(c)). Naturally, GSC and OSC achieve the best compression ratio, since they do not consider temporal information. The result also verifies our previous lemma that GSC and OSC have the same power in terms of compression ratio. Moremore, the compression ratio of OSTC and GSTC grows faster than that of PRESS and NDP, showing more benefits as ϵ_s increases. OSTC achieves the best compression ratio amongst all spatio-temporal compression methods, confirming the optimality of our proposed algorithm. In terms of running time in Figure 9(b) and 9(d), NDP is fastest and almost not affected by ϵ_s, while OSTC is most time-consuming. The running time of our proposed algorithms increase since a greater $|\epsilon_s|$ results in more MRT enumerations during the compression. PRESS is also affected by $|\epsilon_s|$ since it is related to the angular search region during bounded temporal compression. Moreover, GSTC (OSTC) runs slower than GSC (OSC) because of the extra time cost for obtaining the optimal MRT.

Effect of ϵ_t. Obviously, as depicted by Figure 10(a) and 10(c), NDP, GSC and OSC are not affected by ϵ_t since they do not consider temporal information at all. For GSTC and OSTC, a smaller ϵ_t means more time stamps of the MRTs are likely to violate the temporal constraint, leading to more time correction cost, which explains the increasing trend of compression ratio as ϵ_t grows. In addition, the compression ratios of OSTC and GSTC are very close and both outperform PRESS and NDP constantly by a large margin.

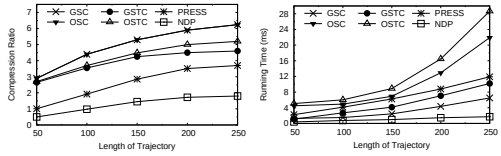


(a) Compression Ratio (BJ) (b) Running Time (BJ)

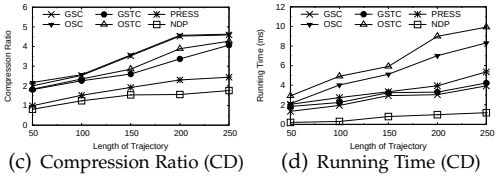


(c) Compression Ratio (CD) (d) Running Time (CD)

Fig. 10. Performance of Compression Algorithms: Effect of ϵ_t



(a) Compression Ratio (BJ) (b) Running Time (BJ)



(c) Compression Ratio (CD) (d) Running Time (CD)

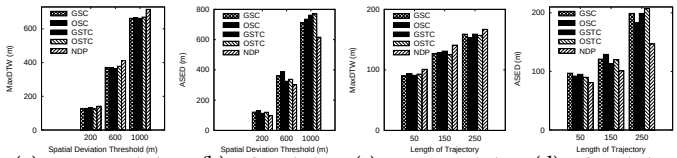
Fig. 11. Performance of Compression Algorithms: Effect of $|T|$

When it comes to efficiency in Figure 10(b) and 10(d), none of the approaches except PRESS is affected by ϵ_t, which is natural for GSC, OSC and NDP. As to GSTC and OSTC, ϵ_t affects the number of time stamps to be rectified but the total number of time stamps to be checked remains the same. The efficiency of PRESS slightly decreases with ϵ_t as its angular search region increases.

Effect of $|T|$. To study the effect of the length of trajectory, we select five groups of trajectories from the test dataset, each including 10000 trajectories with about 50, 100, 150, 200, 250 samples respectively and record the average compression ratio and running time for each group. In Figure 11(a) and 11(c), the compression ratios of all methods increase with $|T|$, as longer trajectories tend to have more redundant samples hence there are more room to improve the compression ratio. Regardless of $|T|$, proposed methods of REST framework well outperform their competitors constantly. As illustrated in Figure 11(b) and 11(d), the running time of all methods increase with the length of trajectory, while the growth of computational cost for OSTC and OSC is relatively faster due to the quadratic complexity with respect to $|T|$ when deriving the optimal storage cost based on dynamic programming.

Reconstruction Error Analysis. Furthermore, to evaluate the accuracy of trajectory compression, we adopt two measures, MaxDTW-based and Average Synchronous Euclidean Distance [22] (ASED)-based spatial error, which are respectively quantified by the MaxDTW distance and average Euclidean distance between a raw trajectory and its corresponding decompressed trajectory. The reason we adopt ASED is that the decompressed trajectory may have

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2914449, IEEE Transactions on Knowledge and Data Engineering



(a) MaxDTW (BJ) (b) ASED (BJ) (c) MaxDTW (BJ) (d) ASED (BJ)

Fig. 12. Spatial Error of Compression Algorithms

different length with its raw correspondence. Due to space limit, we only show the experimental results of BJ dataset. Since PRESS is spatially lossless, we only compare our compression algorithms and NDP by varying ϵ_s and $|T|$. Figure 12(a) shows that all our methods have the similar performances with respect to ϵ_s. This is expected as these methods apply the same error metric (i.e., MaxDTW) during spatial compression. Besides, the spatial error of NDP is higher than that of our methods, demonstrating the superiority of our methods. In terms of ASED-based spatial error in Figure 12(b), NDP performs better than our methods because it adopts synchronous Euclidean distance when compressing trajectories. From Figure 12(c) and 12(d) we can see that the spatial errors of all approaches have a growing tendency as $|T|$ is enlarged since a longer trajectory has more chance to deviate from its corresponding decompressed trajectory.

7.2.3 Performance of Query Processing

In the final set of experiments, we evaluate the effectiveness (*memory cost of index*) and efficiency (*running time*) of the following methods for processing range queries by varying the number of trajectories (to be queried) $|D|$ on BJ dataset, the size of query window $|Q_w|$ and compression ratio CR:

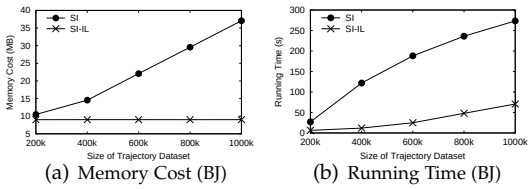
1) Range query processing on raw trajectories with spatial index is denoted as *SI*. This approach of query processing is creating spatial index (i.e., R-tree) on the original trajectories to speed up the query processing.

2) Range query processing on partially decompressed trajectory data is denoted as *SI-IL*. This method takes IR-tree as the spatial index for compressed trajectories.

In order to compare the performance of the above two approaches for range query processing, we randomly generate 10^3 queries on test dataset (i.e., the trajectory set to be queried) and record the total running time. These queries are process in the raw trajectories and compressed trajectories (generated by OSTC) respectively.

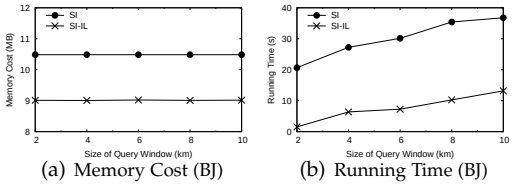
Effect of $|D|$. To study the scalability of the algorithms, we generate 5 datasets containing from $200k$ to $1000k$ trajectories by random selection from the original trajectory set. Then we run experiments on these five datasets, the results of which are depicted in Figure 13. As shown in Figure 13(a), the memory cost of SI increases as the size of trajectory dataset is enlarged since its index is mainly established on these trajectories, while SI-IL keeps a constant memory cost when enlarging $|D|$ due to the fact that its index is mainly established with the reference set whose size is stable. When referring to the running time in Figure 13(b), as expected, though the running time increase as the number of trajectories increases, our proposed algorithm scales well with the size of the dataset generally.

Effect of $|Q_w|$. Subsequently, we study the effects of the size of query window $|Q_w|$, the radius of circular query window, by changing it from 2km to 10km. Not surprisingly, as we can see from Figure 14, SI-IL significantly outperforms SI for all values of $|Q_w|$ in terms of both memory cost and runtime. For memory cost, SI and SI-IL keep a constant cost as $|Q_w|$ increases since the number of trajectories used to construct the index is stable, which is demonstrated in Figure 14(a). For efficiency, the performances of both algorithms deteriorate with the increasing size of query window (see Figure 14(b)). This is because statistically the relevance of a trajectory will be affected by more points as $|Q_w|$ increases. In other words, more nodes of the R-tree overlap with the larger query window and need to be examined. Besides, we also notice that the running time of SI-IL is far less than that of SI, which testifies the superiority of SI-IL.



(a) Memory Cost (BJ) (b) Running Time (BJ)

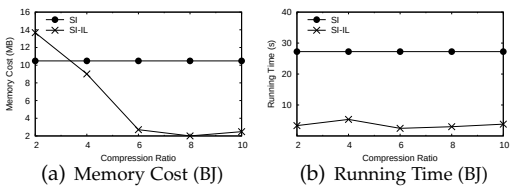
Fig. 13. Performance of Query Processing: Effect of $|D|$



(a) Memory Cost (BJ) (b) Running Time (BJ)

Fig. 14. Performance of Query Processing: Effect of $|Q_w|$

Effect of CR. In the final experiments, we investigate how the compression ratio CR affects the performance of our proposed SI-IL method. We choose 5 groups of trajectories whose compression ratios are approximately equal to 2, 4, 6, 8, 10 respectively. Generally, a large CR means that most part of the raw trajectory can be compressed by the reference (sub-)trajectories while a small CR means more original samples of the trajectory are kept for representing it when being compressed. From Figure 15(a) we can see that, SI-IL approach performs badly when CR is small (i.e., $CR < 3.5$), which consumes more memory than SI, since SI-IL has to construct a complicated index structure for a large number of original samples of the trajectories that are used to represent the trajectories. However, when $CR \geq 6$, the memory cost remains relatively small and unchanged. This may be due to the fact that the raw trajectories can be almost compressed by the reference (sub-)trajectories and the index can be only constructed on the reference (sub-)trajectories. Besides, Figure 15(b) illustrates SI-IL is consistently more efficient than SI, showing the benefits of our proposed method.



(a) Memory Cost (BJ) (b) Running Time (BJ)

Fig. 15. Performance of Query Processing: Effect of CR

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TKDE.2019.2914449, IEEE Transactions on Knowledge and Data Engineering

8 Related Work

8.1 Trajectory Compression

The existing trajectory compression algorithms can be classified into two categories: 1) spatial compression; 2) spatio-temporal compression.

Spatial compression algorithms treat trajectories as polylines. For example, Douglas-Peucker (DP) algorithm [7] is a classic line generalization approach that uses a perpendicular distance threshold to reduce the number of points. As DP algorithm is simple and feasible, a variety of applications have been proposed [21] to speed up DP algorithm. Bellman's algorithm [2] fits a finite number of line segments to a curve based on dynamic programming, preserving the most essential spatial features. Due to the pure geometric nature, the above algorithms cannot be applied when the temporal information of trajectories also matters.

Taking the temporal component of trajectories into account, Sliding Window Algorithm [12] and Opening Window Algorithm [22], [23] are designed to keep spatio-temporal information of a trajectory within a sliding window to compress it. Muckell et al. introduce a heuristic method called SQUISH [24], using a priority queue where the priority of each point is defined as an estimate of the error that the removal of that point would introduce, to compress trajectories. Recently, a lossless path compressor in road network is developed in [11], namely Minimum Entropy Labeling, which guarantees a theoretic bound and achieves practically high spatio-temporal compressibility. [13] presents a compressed data structure for moving object trajectories, where the compressed trajectories can be represented as sequences of road edges. Song et al. develop a framework, PRESS [29], that separates a given trajectory in a road network into spatial path and time sequence components. These two components are then compressed by Hybrid Spatial Compression algorithm and error Bounded Temporal Compression algorithm respectively, achieving spatial lossless and temporal error-bounded compression. However, the pre-computation and storage of all-pair shortest paths and most frequent paths require a stable road network and large memory space to be available, which limits its applied scenarios.

8.2 Trajectory Index and Query

As a fundamental trajectory data manipulation, trajectory query is becoming crucial. Building efficient spatio-temporal index structures on trajectories can significantly facilitate query processing in trajectory databases. Koide et al. propose a novel spatio-temporal index structure for Network-Constrained Trajectories (NCTs), namely Suffix-array-based Network-constrained Trajectory index [14]. Then they further design a compressed-index method for NCTs by converting NCTs into a trajectory string, in which Relative Movement Labeling and FM-index are incorporated to accelerate the query processing [13]. However, the above existing work targets network-constrained data and uses this characteristic to improve the efficiency of query processing and the accuracy of the query results. Effective index structures [16], [26], [33], [34] are built to manage trajectories in Euclidean space and support high performance trajectory queries, among which R-tree [10] is the most common index adopted to accelerate the query processing. Here, we also process the trajectory query over an R-tree index of all the reference (sub-)trajectories. Notice that we do not consider the STR-tree or TB-tree [26] for trajectory index since they focus more on trajectory preservation and leave other spatial properties like spatial proximity aside, while in the our problem, the R-tree index is only for fast retrieval of nearest trajectories.

For trajectory query, a typical one asks for the information against spatio-temporal relationships between trajectories and other spatial data objects, i.e. points, regions and trajectories [6]. For instance, Chen et al. propose k-Path Nearest Neighbor (k-PNN) query to return the k-NN with respect to shortest path connecting a given point (i.e., a destination); SETI [4] and PA-tree [25] are designed for range queries which seek to find all trajectories that intersect a spatial region; and SECONDO [9] and TrajStore [5] are developed for performing regular k-NN queries on trajectories using the Euclidean distance.

9 Conclusion and Future Work

In this paper we propose a novel data-driven framework, called REST, to compress the spatio-temporal information of trajectory data. In order to achieve high effectiveness and efficiency, we addressed a few challenges by proposing different strategies to construct a compact but expressive reference set, and designing efficient and optimal algorithms to represent a given trajectory with selected matchable reference trajectories. To the best our knowledge, it is the first data-drive approach to compress trajectories in unconstrained space with both spatial and temporal dimensions considered. In addition, as the compressed trajectories are in the form of sequence of reference trajectories and original samples, we develop an effective index structure to support efficient query processing over compressed trajectories without full decompression. Extensive empirical study based on real trajectories dataset also confirms the superiority of our proposed framework over the state-of-the-art approaches in terms of compression ratio, efficiency and space cost.

Acknowledgements

The work is supported by the National Natural Science Foundation of China (Grant No. 61532018, 61836007, 61832017, 61772356 and 61872258), Australian Research Council (Grants No. DP170101172) and Major Project of Zhejiang Lab (No. 2019DH0ZX01).

References

[1] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, pages 3–14, 1995.
[2] R. Bellman. On the approximation of curves by line segments using dynamic programming. *Archives of Internal Medicine*, 4(6):284, 1961.
[3] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD*, pages 359–370, 1994.
[4] V. P. Chakka, A. Everspaugh, and J. M. Patel. Indexing large trajectory data sets with seti. In *CIDR*, 2003.
[5] P. Cudre-Mauroux, E. Wu, and S. R. Madden. Trajstore: An adaptive storage system for very large trajectory data sets. In *ICDE*, 2010.

[6] K. Deng, K. Xie, K. Zheng, and X. Zhou. *Trajectory indexing and retrieval*. Computing with Spatial Trajectories, 2011.

[7] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *International Journal for Geographic Information and Geovisualization*, 10(2):112–122, 1973.

[8] M. C. González, C. A. Hidalgo, and A.-L. Barabási. Understanding individual human mobility patterns. *Nature*, 453(7196):779–782, 2008.

[9] R. H. Guting, A. Braese, T. Behr, and J. Xu. Nearest neighbor search on moving object trajectories in secondo. In *SSTD*, 2009.

[10] A. Guttman. R-trees: a dynamic index structure for spatial searching. In *SIGMOD*, pages 47–57, 1984.

[11] Y. Han, W. Sun, and B. Zheng. Compress: a comprehensive framework of trajectory compression in road networks. *TODS*, 42(2):1–49, 2017.

[12] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *ICDM*, pages 289–296, 2001.

[13] S. Koide, Y. Tadokoro, C. Xiao, and Y. Ishikawa. Cinct: Compression and retrieval for massive vehicular trajectories via relative movement labeling. In *ICDE*, pages 1097–1108, 2018.

[14] S. Koide, Y. Tadokoro, and T. Yoshimura. Snt-index: Spatio-temporal index for vehicular trajectories on a road network based on substring matching. In *SIGSPATIAL*, pages 1–8, 2015.

[15] R. Lange, F. Dürr, and K. Rothermel. Efficient real-time trajectory tracking. *VLDBJ*, 20(5):671–694, 2011.

[16] C. Lei and R. Ng. On the marriage of lp-norms and edit distance. In *VLDB*, pages 792–803, 2004.

[17] C. Lei, M. T. Ozsu, and V. Oria. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.

[18] J. Li, J. Wang, L. Yu, and J. Zhang. A novel frequent trajectory mining method based on gsp. In *WISM*, pages 134–140, 2011.

[19] Z. Li, K. C. Lee, B. Zheng, W. Lee, D. L. Lee, and X. Wang. Ir-tree: An efficient index for geographic document search. *TKDE*, 23(4):585–599, 2011.

[20] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, and R. Jurdak. Bounded quadrant system: Error-bounded trajectory compression on the go. In *ICDE*, pages 987–998, 2015.

[21] R. B. McMaster. A statistical analysis of mathematical measures for linear simplification. *The American Cartographer*, 13(2):103–116, 1986.

[22] N. Meratnia and A. Rolf. Spatiotemporal compression techniques for moving point objects. In *EDBT*, pages 765–782, 2004.

[23] J. Muckell, J. H. Hwang, C. T. Lawson, and S. S. Ravi. Algorithms for compressing gps trajectory data: an empirical evaluation. In *SIGSPATIAL*, pages 402–405, 2010.

[24] J. Muckell, J.-H. Hwang, V. Patil, C. T. Lawson, F. Ping, and S. Ravi. Squish: an online approach for gps trajectory compression. In *COM.Geo*, page 13, 2011.

[25] J. Ni and C. V. Ravishankar. Pa-tree: A parametric indexing scheme for spatio-temporal trajectories. In *SSTD*, pages 254–272, 2005.

[26] D. Pfoser, C. S. Jensen, and Y. Theodoridis. Novel approaches to the indexing of moving object trajectories. *VLDB*, pages 395–406, 2000.

[27] M. Potamias, K. Patroumpas, and T. Sellis. Sampling trajectory streams with spatiotemporal criteria. In *SSDBM*, pages 275–284, 2006.

[28] S. Ranu, D. P, A. D. Telang, P. Deshpande, and S. Raghavan. Indexing and matching trajectories under inconsistent sampling rates. In *ICDE*, pages 999–1010, 2015.

[29] R. Song, W. Sun, B. Zheng, and Y. Zheng. Press: a novel framework of trajectory compression in road networks. *VLDB Endowment*, 7(9):661–672, 2014.

[30] G. Trajcevski, H. Cao, P. Scheuermanny, O. Wolfsonz, and D. Vaccaro. On-line data reduction and the quality of history in moving objects databases. In *MobiDE*, pages 19–26, 2006.

[31] M. Vlachos, D. Gunopoulos, and G. Kollios. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, 2002.

[32] Y. Zhao, S. Shang, Y. Wang, B. Zheng, Q. V. H. Nguyen, and K. Zheng. Rest: a reference-based framework for spatio-temporal trajectory compression. In *SIGKDD*, pages 2797–2806, 2018.

[33] B. Zheng, H. Wang, K. Zheng, H. Su, K. Liu, and S. Shang. Sharkdb: an in-memory column-oriented storage for trajectory analysis. *WWWJ*, 21(2):1–31, 2017.

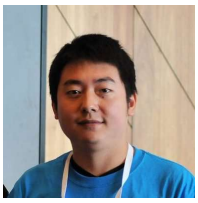
[34] K. Zheng, S. Shang, N. J. Yuan, and Y. Yang. Towards efficient search for activity trajectories. In *ICDE*, pages 230–241, 2013.

[35] K. Zheng, Y. Zheng, X. Xie, and X. Zhou. Reducing uncertainty of low-sampling-rate trajectories. In *ICDE*, pages 1144–1155, 2012.

Kai Zheng is a Professor of Computer Science with University of Electronic Science and Technology of China. He received his PhD degree in Computer Science from The University of Queensland in 2012. He has been working in the area of spatial-temporal databases, uncertain databases, social-media analysis, in-memory computing and blockchain technologies. He has published over 100 papers in prestigious journals and conferences in data management field such as SIGMOD, ICDE, VLDB Journal, ACM Transactions and IEEE Transactions. He is a member of IEEE.

Yan Zhao received the Master degree in Geographic Information System from University of Chinese Academy of Sciences, in 2015. She is currently a PHD student in Soochow University. Her research interests include spatial database and trajectory computing.

Defu Lian received the BE and PhD degrees in computer science from the University of Science and Technology of China (USTC), in 2009 and 2014, respectively. He is currently a research professor with the University of Science and Technology of China. His main research interests include mobile data mining and recommender systems. He has published more than 40 papers in journals and conferences such as KDD, WWW, IJCAI, and ACM TOIS, IEEE TDKE. He also was a reviewer for several journals such as the TKDE, TOIS and KAIS.



Bolong Zheng is currently working as an associate professor in Huazhong University of Science and Technology. He received the PhD degree from the University of Queensland, in 2017. After that, he worked as a postdoctoral research fellow with the University of Queensland and Aalborg University. His research interests include trajectory querying and mining, social-media analysis, spatial-temporal databases.

Guanfeng Liu is current a Lecturer in the Department of Computing at Macquarie University, Australia. He received his Ph.D degree in Computer Science from Macquarie University, Australia in 2013. His research interests include graph data management, trust computing and social networks. He has published over 60 papers in the most prestigious journals and conferences such as IJCAI, AAAI, ICDE, CIKM, TKDE, TSC and ICWS.

Xiaofang Zhou received the bachelor's and master's degrees in computer science from Nanjing University, in 1984 and 1987, respectively, and the PhD degree in computer science from the University of Queensland in 1994. He is a professor of computer science with the University of Queensland. He is the head of the Data and Knowledge Engineering Research Division, School of Information Technology and Electrical Engineering. He is also a specially appointed adjunct professor with Soochow University, China. His research is focused on finding effective and efficient solutions to managing integrating, and analyzing very large amounts of complex data for business and scientific applications. His research interests include spatial and multimedia databases, high performance query processing, web information systems, data mining, and data quality management. He is a fellow of IEEE.