REST: A Reference-based Framework for Spatio-temporal Trajectory Compression

Yan Zhao
Institute of Artificial Intelligence & School of Computer Science and Technology, Soochow University
Suzhou, China
zhaoyan@suda.edu.cn

Shuo Shang*
King Abdullah University of Science and Technology
Thuwal, Saudi Arabia
jedi.shang@gmail.com

Yu Wang
The Chinese University of Hong Kong
Hong Kong, China
yuwang@cse.cuhk.edu.hk

Bolong Zheng
Huazhong University of Science and Technology
Aalborg University
bolongzheng@hust.edu.cn

Quoc Viet Hung Nguyen
Griffith University
Brisbane, Australia
quocviethung.nguyen@griffith.edu.au

Kai Zheng*
University of Electronic Science and Technology of China
Chengdu, China
zhengkai@uestc.edu.cn

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. 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.

*Corresponding authors: Kai Zheng and Shuo Shang.

KDD '18, August 19–23, 2018, London, United Kingdom

ACM ISBN 978-1-4503-5552-0/18/08…$15.00
https://doi.org/10.1145/3219819.3220030

CCS CONCEPTS

• **Information systems** → **Spatial-temporal systems**; *Data compression*; *Location based services*;

KEYWORDS

compression algorithm, trajectory, spatio-temporal data

ACM Reference Format:
Yan Zhao, Shuo Shang, Yu Wang, Bolong Zheng, Quoc Viet Hung Nguyen, and Kai Zheng*. 2018. REST: A Reference-based Framework for Spatio-temporal Trajectory Compression. In *KDD '18: The 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, August 19–23, 2018, London, United Kingdom,* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, Article 4, 10 pages. https://doi.org/10.1145/3219819.3220030

1 INTRODUCTION

The prevalence of various mobile devices has resulted in a massive amount of trajectory data. While they contain a wealth of mobility information [28, 29, 33] 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 [6, 19, 32]. 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.

Trajectory compression approaches can be generally classified into spatial and spatio-temporal compression. Treating trajectories as polylines, spatial compression methods are also known as line simplification algorithms [2, 3, 7], 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 [14] and location tracking [16]. 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, 19, 21, 25, 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. [27] 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 range of free-space moving objects such as animals, flying objects 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 [34] (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.

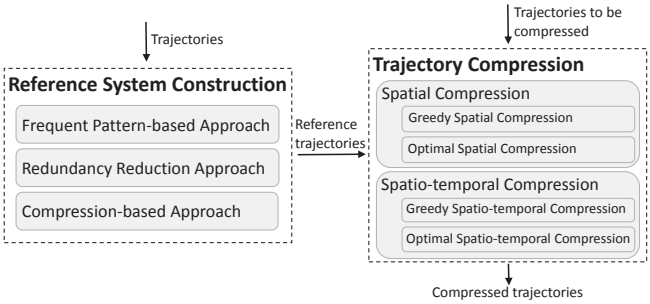


Figure 1: REST Framework Overview

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 large training dataset. 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.

Our main contributions can be summarized as follows:

- We propose a data-driven trajectory compression framework targeting free-space trajectory data and considering spatio-temporal dimensions.
- We present three strategies to build a reference trajectory set with high coverage and low redundancy.
- We propose greedy and optimal algorithms for both spatial-only and spatio-temporal compression to trade off compression efficiency and effectiveness.
- We conduct extensive experiments based on real taxi trajectory dataset, which empirically demonstrate the advantages of our proposed framework compared to the representative approaches.

The remainder of this paper is organized as follows. Section 2 introduces the preliminary concepts, error metric and reference-based compression problem. We then propose the construction of reference trajectories in Section 3. The algorithms for spatial-only and spatio-temporal compression are presented in Section 4 and Section 5 respectively. We report the results from empirical study in Section 6, followed by the related work in Section 7. Section 8 concludes this paper and outlines the direction to further extend our work.

[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.

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.

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:

- *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., $CR = \frac{space(T)}{space(T')}$.
- *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 work, we propose a simple but effective variant of Dynamic Time Warping (DTW) [4] 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:*

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

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

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

Similar to DTW, we can use a dynamic programming algorithm [4] to compute MaxDTW.

2.3 Reference-based Compression

As observed in previous studies [34], 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.

An ideal case is that the trajectory T can be fully represented by selected trajectories in R, so the compressed trajectory T' will be in the form of sub-trajectory sequence, i.e., $T' = ((\mathbb{T}_1, s_1, e_1), (\mathbb{T}_2, s_2, e_2), ...)$, where $\mathbb{T}_i \in R$ (recorded by a 4-byte identifier), s_i (e_i) is the start (end) index with respect to $\mathbb{T}_i$ (each recorded by a 2-byte integer since a trajectory cannot be quite long). Therefore each triplet of T' only occupies 8 bytes in memory, the same as that of an original sample point when temporal information is not considered. If a raw trajectory with 100 sample points can be represented by 10 reference trajectories, the compression ratio will be $800/80 = 10$.

Inevitably, with the constraint of ϵ there may exist some parts that cannot be described by any reference trajectory. In such cases we simply keep its original samples for that part in the compressed form. So in practice a compressed trajectory T' is in the mixture form of sample sequence and sub-trajectory sequence. The first bit of each triplet can be spared to indicate if the next 8-byte space holds an

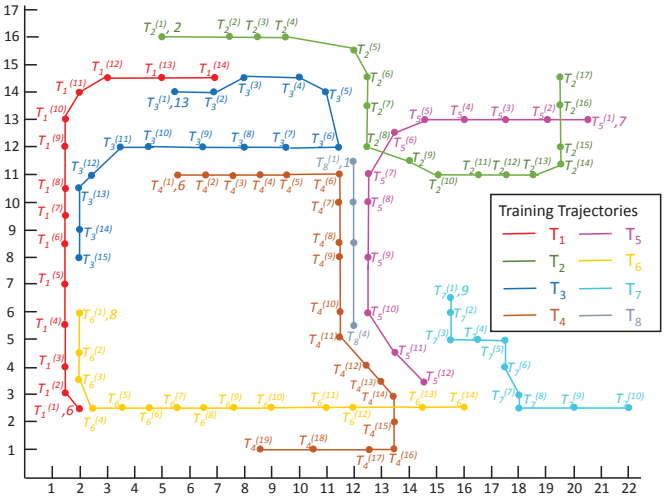


Figure 2: Running example

original sample point or a reference trajectory triplet without incurring any space overhead.

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 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 [15], 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:

- Find all calculating points and calculating trajectories.
- Given the minimum support threshold, the frequent calculating points are obtained by scanning all calculating points.
- Remove non-frequent calculating points, and obtain frequent calculating sub-trajectories.
- Remove the frequent calculating sub-trajectories who is the sub-trajectories of another frequent calculating trajectories.
- 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. To save space, we omit the pseudo-code.

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.,*

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

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 ($2 < n \le |T|$) 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, $M(T^{(2,9)}) = \{T_2^{(1,8)}\}$, $M(T^{(10,15)}) = \{T_4^{(6,14)}, T_5^{(7,12)}\}$.

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.

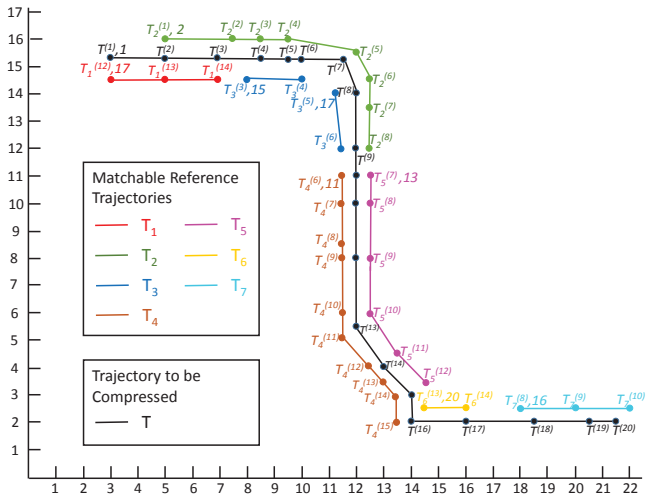


Figure 3: Matchable Reference Trajectories

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** $MaxDTW(T^{(i,j)}, \mathbb{T}_a^{(m,n)}) \le \epsilon_s$ **then**
7 Add $\mathbb{T}_a^{(m,n)}$ into $M(T^{(i,j)})$;
8 **if** $MaxDTW(T^{(i,j)}, \mathbb{T}_b^{(s,t)}) \le \epsilon_s$ **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).

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

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 $1 \le j < i$; 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 $j = i$.

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';
```

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 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 $O(|T|^2)$.

Take 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) $(n \le |T_a + T_b - 1|)$. 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 $(a_i, b_i)(1 \le i \le n)$, and performing the following actions:

- Initialize $t = 0$ to record the most recent correct timestamp of T_b and set $b_0.t = 0$, $C_{T_a}^{\epsilon_t}(T_b) = 0$;
- If $|t + b_i.t - b_{i-1}.t - a_i.t| \le \epsilon_t$, update $t = t + b_i.t - b_{i-1}.t$;
- 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 $b_i^{new}.t = b_{i-1}^{new}.t + b_i^{old}.t - b_{i-1}^{old}.t$.

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

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.

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

$F_T[i]$ 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 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.

Table 3: Experiment Parameters

Parameters	Values
No. of trajectories to construct reference set $\|D_R\|$	50k, <u>100k</u>, 150k, 200k, 250k
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 query trajectory $\|T\|$	50, 100, 150, 200, 250

6 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.

6.1 Experiment Setup

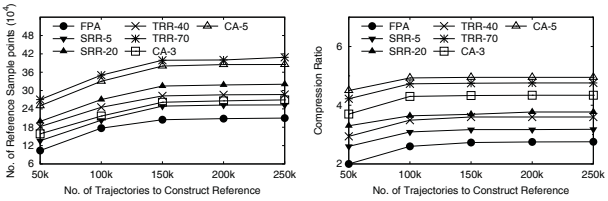
We use a real trajectory dataset generated by taxis in Beijing over one week, which contains about 2 million trajectories. Trajectories in one day (about 280k) are used as training dataset to construct the reference set, and the rest are used to test the performance of different compression algorithms 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.

6.2 Experiment Results

6.2.1 Performance of Reference Set Construction. We first evaluate the performance of reference set construction and its impact to spatial compression. Two metrics, *Size of Reference Set* (the number of sampling points in the reference set) and *Compression Ratio*, are compared among the following methods (specified in Section 3) by varying $|D_R|$, ϵ_s.

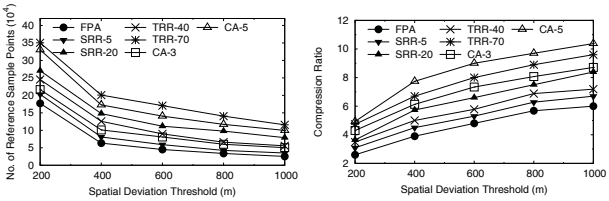
- FPA: Frequent Pattern-based Approach with minimum support 100.
- SRR-5: Segment Redundancy Reduction approach with minimum sub-trajectory length $\eta = 5$.
- SRR-20: Segment Redundancy Reduction approach with minimum sub-trajectory length $\eta = 20$.
- TRR-40: Trajectory Redundancy Reduction approach with overlap threshold $\theta = 40\%$.
- TRR-70: Trajectory Redundancy Reduction approach with overlap threshold $\theta = 70\%$.
- CA-3: Compression-based Approach with compression ratio threshold $\delta = 3$.
- CA-5: Compression-based Approach with compression ratio threshold $\delta = 5$.

Effect of $|D_R|$. In this set of experiment, we study the effect of $|D_R|$. As shown in Figure 4(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



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

Figure 4: Performance of Reference Set Construction: Effect of $|D_R|$



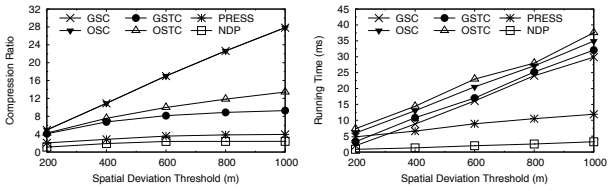
(a) Size of Reference Set (b) Compression Ratio

Figure 5: Performance of Reference Set Construction: Effect of $|\epsilon_s|$

is found that the size of reference set almost stops growing beyond 400k, which only takes a few megabytes memory space. From Figure 4(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.

Effect of ϵ_s. Next we study the effect of spatial deviation threshold. In Figure 5(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 5(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.

6.2.2 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. Moreover, we also implement two representative spatio-temporal compression algorithms, namely Normal Douglas-Peucker



(a) Compression Ratio (b) Running Time

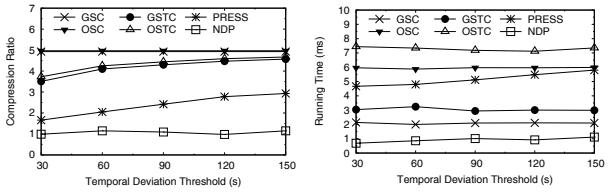
Figure 6: Performance of Compression Algorithms: Effect of $|\epsilon_s|$

(NDP) algorithm based on SED [19] (a spatial trajectory simplification algorithm with synchronous Euclidean distance) and PRESS [27] (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 $|\epsilon_s|$. As expected, compression ratio of all algorithms gradually increases as ϵ_s grows (see Figure 6(a)). 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 6(b), 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.

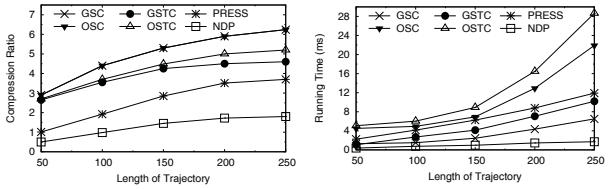
Effect of ϵ_t. Obviously, as depicted by Figure 7(a), NDP, GSC and OSC are not affected by ϵ_t since they do not consider temporal information at all. For GSTC and OSTC, a smaller ϵ_t means more time stamps of the MRTs are likely to violate the temporal constraint, leading to more time correction cost, 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. When it comes to efficiency in Figure 7(b), 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,



(a) Compression Ratio (b) Running Time

Figure 7: Performance of Compression Algorithms: Effect of $|\epsilon_t|$



(a) Compression Ratio (b) Running Time

Figure 8: Performance of Compression Algorithms: Effect of $|T|$

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 8(a), 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 8(b), 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.

Summary of our empirical study:

- The generated reference set is very space efficient.
- OSTC has the best compression ratio but sacrifices some efficiency.
- GSTC achieves good balance between efficiency (second to NDP) and effectiveness (second to OSTC).

7 RELATED WORK

The existing trajectory compression algorithms can be classified into two categories based on whether taking the temporal information into account: 1) spatial compression algorithms [2, 3, 7, 10, 13, 18, 31]; 2) spatio-temporal compression algorithms [9, 12, 19–22, 24, 25].

7.1 Spatial Compression Algorithms

Spatial compression algorithms treat the trajectories as polylines, so they are sometimes referred to as line generalization/simplification algorithms. Douglas-Peucker (DP) algorithm [7] is a classic line generalization approach that uses a perpendicular distance threshold to reduce the number of points while preserves directional trends of lines when displaying geographic features. As DP algorithm is simple and feasible, a variety of applications have been proposed [18]. Later Hershberger et al. [10] speed up DP algorithm to the time complexity of O($nlogn$), in which n is the number of original data points. Similar to the DP algorithm, Bellman's algorithm [2, 3] also fits a finite number of line segments to a curve using dynamic programming and obtains the optimal solution by minimizing the root mean square error yet preserving the most essential spatial features. The Reservoir Sampling Algorithm [31] is proposed to select a random sample of n points at equal probability without replacement from a pool of N ($n \leq N$) records, in which the value of N is unknown in advance. Due to the pure geometric nature, the above algorithms cannot be applied when the temporal information of trajectories also matters.

7.2 Spatio-temportal Compression Algorithms

Taking the temporal component of trajectories into account, Meratnia et al. [19] propose an algorithm, called TD-TR, for the purpose of sampling and dilution over trajectory streams by modifying the distance formula of DP algorithm. The Sliding Window Algorithm [12] is designed to keep spatio-temporal information of a trajectory within a sliding window by taking the first point of the potential line segment as an anchor and then growing the segment until it exceeds some error bounds. Such process repeats with the next data point until the entire positional time series has been simplified into a piecewise line. Inspired by Sliding Window Algorithm, Opening Window Algorithm [19, 20] also anchors the first point in the line series, but defines the third point as float in the series. The float slides forward to each subsequent data point of the trajectory until the distance threshold is violated or the float reaches the end of the trajectory. Once the violation occurs, the end of the current line segment, which is also the anchor of the next segment, has two choices: 1) the point resulting in the violation (Normal Opening Window Algorithm); 2) the point before the one that causes the violation (Before Opening Window Algorithm) [20]. Muckell et al. introduce a heuristic method called Spatial Quality Simplification Heuristic (SQUISH) [21], 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. SQUISH compresses each trajectory by removing points of the lowest priority from the priority queue until it achieves the target compression ratio. Considering the spatio-temporal information as well as the sequential nature of the trajectory data, STTrace [25] algorithm is developed for trajectory compression to catch the significant spatio-temporal features in a trace, such as direction and speed. All the above work only utilize the spatio-temporal characteristics of the trajectory to be compressed. Recently, Song et al. develop a framework, PRESS [27], that leverages the constraint of road network to achieve better compression performance. To adopt this

framework, a digital map with topological structure of the road network is required such that the target trajectory can be transformed into a sequence of road segment using map matching algorithms [5, 11, 17, 23, 26]. PRESS separates the spatio-temporal information of a given trajectory into spatial path and time sequence, which are then compressed by Hybrid Spatial Compression (HSC) algorithm and error Bounded Temporal Compression (BTC) 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 the applied scenarios of PRESS.

8 CONCLUSION AND FUTURE WORK

In this paper we propose a novel data-driven framework, called REST, to compress the spatio-temporal trajectories. 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. Extensive empirical study based on real trajectories dataset confirms the superiority of our proposed framework over the state-of-the-art approaches in terms of compression ratio, efficiency and space cost. Since the compressed trajectories are in the form of sequence of reference trajectories, our next step is to develop effective indexing structures to support efficient query processing over compressed trajectories without full decompression.

ACKNOWLEDGMENTS

This research is partially supported by the Natural Science Foundation of China (Grant No. 61532018, 61502324).

REFERENCES

[1] Rakesh Agrawal and Ramakrishnan Srikant. 1995. Mining sequential patterns. In *ICDE*. 3–14.
[2] Richard Bellman. 1961. On the approximation of curves by line segments using dynamic programming. *Archives of Internal Medicine* 4, 6 (1961), 284.
[3] Richard E Bellman. 1962. Applied dynamic programming. *Princeton University Press* (1962).
[4] Donald J Berndt and James Clifford. 1994. Using dynamic time warping to find patterns in time series. In *KDD*, Vol. 10. 359–370.
[5] Sotiris Brakatsoulas, Dieter Pfoser, Randall Salas, and Carola Wenk. 2005. On map-matching vehicle tracking data. In *VLDB*. 853–864.
[6] Hu Cao, Ouri Wolfson, and Goce Trajcevski. 2006. Spatio-temporal data reduction with deterministic error bounds. *VLDBJ* 15, 3 (2006), 211–228.
[7] David H Douglas and Thomas K Peucker. 1973. 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 (1973), 112–122.
[8] Marta C González, César A Hidalgo, and Albert-László Barabási. 2008. Understanding individual human mobility patterns. *Nature* 453, 7196 (2008), 779–782.
[9] Yunheng Han, Weiwei Sun, and Baihua Zheng. 2017. COMPRESS: a comprehensive framework of trajectory compression in road networks. *TODS* 42, 2 (2017), 1–49.
[10] John Hershberger and Jack Snoeyink. 1992. Speeding Up the Douglas-Peucker Line-Simplification Algorithm. *Proc.intl.symp.on Spatial Data Handling* (1992), 134–143.
[11] Georgios Kellaris, Nikos Pelekis, and Yannis Theodoridis. 2013. Map-matched trajectory compression. *Journal of Systems & Software* 86, 6 (2013), 1566–1579.
[12] Eamonn Keogh, Selina Chu, David Hart, and Michael Pazzani. 2001. An online algorithm for segmenting time series. In *ICDM*. 289–296.
[13] Benjamin Krogh, Christian S Jensen, and Kristian Torp. 2016. Efficient in-memory indexing of network-constrained trajectories. In *SIGSPATIAL*. 1–10.
[14] Ralph Lange, Frank Dürr, and Kurt Rothermel. 2011. Efficient real-time trajectory tracking. *VLDBJ* 20, 5 (2011), 671–694.
[15] Junhuai Li, Jinqin Wang, Lei Yu, and Jing Zhang. 2011. A Novel Frequent Trajectory Mining Method Based on GSP. In *WISM*. 134–140.
[16] Jiajun Liu, Kun Zhao, Philipp Sommer, Shuo Shang, Brano Kusy, and Raja Jurdak. 2015. Bounded quadrant system: Error-bounded trajectory compression on the go. In *ICDE*. 987–998.
[17] Yin Lou, Chengyang Zhang, Yu Zheng, Xing Xie, Wei Wang, and Yan Huang. 2009. Map-matching for low-sampling-rate GPS trajectories. In *SIGSPATIAL*. 352–361.
[18] Robert B McMaster. 1986. A statistical analysis of mathematical measures for linear simplification. *The American Cartographer* 13, 2 (1986), 103–116.
[19] Nirvana Meratnia and A Rolf. 2004. Spatiotemporal compression techniques for moving point objects. In *EDBT*. 765–782.
[20] Jonathan Muckell, Jeong Hyon Hwang, Catherine T Lawson, and S. S Ravi. 2010. Algorithms for compressing GPS trajectory data: an empirical evaluation. In *SIGSPATIAL*. 402–405.
[21] Jonathan Muckell, Jeong-Hyon Hwang, Vikram Patil, Catherine T Lawson, Fan Ping, and SS Ravi. 2011. SQUISH: an online approach for GPS trajectory compression. In *COM.Geo*. 13.
[22] Jonathan Muckell, Paul W Olsen, Jeong-Hyon Hwang, Catherine T Lawson, and SS Ravi. 2014. Compression of trajectory data: a comprehensive evaluation and new approach. *GeoInformatica* 18, 3 (2014), 435–460.
[23] Paul Newson and John Krumm. 2009. Hidden Markov map matching through noise and sparseness. In *SIGSPATIAL*. 336–343.
[24] Iulian Sandu Popa, Karine Zeitouni, Vincent Oria, and Ahmed Kharrat. 2015. Spatio-temporal compression of trajectories in road networks. *Geoinformatica* 19, 1 (2015), 117–145.
[25] Michalis Potamias, Kostas Patroumpas, and Timos Sellis. 2006. Sampling Trajectory Streams with Spatiotemporal Criteria. In *SSDBM*. 275–284.
[26] Renchu Song, Wei Lu, Weiwei Sun, Yan Huang, and Chunan Chen. 2012. Quick map matching using multi-core CPUs. In *SIGSPATIAL*. 605–608.
[27] Renchu Song, Weiwei Sun, Baihua Zheng, and Yu Zheng. 2014. PRESS: a novel framework of trajectory compression in road networks. *VLDB Endowment* 7, 9 (2014), 661–672.
[28] Han Su, Kai Zheng, Jiamin Huang, Haozhou Wang, and Xiaofang Zhou. 2015. Calibrating trajectory data for spatio-temporal similarity analysis. *VLDBJ* 24, 1 (2015), 93–116.
[29] Han Su, Kai Zheng, Kai Zeng, Jiamin Huang, Shazia Sadiq, Nicholas Jing Yuan, and Xiaofang Zhou. 2015. Making sense of trajectory data: a partition-and-summarization approach. In *ICDE*. 963–974.
[30] Goce Trajcevski, Hu Cao, Peter Scheuermanny, Ouri Wolfsonz, and Dennis Vaccaro. 2006. On-line data reduction and the quality of history in moving objects databases. In *MobiDE*. 19–26.
[31] Jeffrey S Vitter. 1985. Random sampling with a reservoir. *TOMS* 11, 1 (1985), 37–57.
[32] Haozhou Wang, Kai Zheng, Jiajie Xu, Bolong Zheng, Xiaofang Zhou, and Shazia Sadiq. 2014. Sharkdb: An in-memory column-oriented trajectory storage. In *CIKM*. 1409–1418.
[33] Bolong Zheng, Nicholas Jing Yuan, Kai Zheng, Xing Xie, Shazia Sadiq, and Xiaofang Zhou. 2015. Approximate keyword search in semantic trajectory database. In *ICDE*. 975–986.
[34] Kai Zheng, Yu Zheng, Xing Xie, and Xiaofang Zhou. 2012. Reducing uncertainty of low-sampling-rate trajectories. In *ICDE*. 1144–1155.