TRACE: Real-time Compression of Streaming Trajectories in Road Networks

Tianyi Li[§], Lu Chen[†], Christian S. Jensen[§], Torben Bach Pedersen[§]
[§]Department of Computer Science, Aalborg University, Denmark
[†]College of Computer Science, Zhejiang University, Hangzhou, China
[§]{tianyi, csj, tbp}@cs.aau.dk [†]luchen@zju.edu.cn

ABSTRACT

The deployment of vehicle location services generates increasingly massive vehicle trajectory data, which incurs high storage and transmission costs. A range of studies target offline compression to reduce the storage cost. However, to enable online services such as real-time traffic monitoring, it is attractive to also reduce transmission costs by being able to compress streaming trajectories in real-time. Hence, we propose a framework called TRACE that enables compression, transmission, and querying of network-constrained streaming trajectories in a fully online fashion. We propose a compact two-stage representation of streaming trajectories: a speed-based representation removes redundant information, and a multiple-references based referential representation exploits subtrajectory similarities. In addition, the online referential representation is extended with reference selection, deletion and rewriting functions that further improve the compression performance. An efficient data transmission scheme is provided for achieving low transmission overhead. Finally, indexing and filtering techniques support efficient real-time range queries over compressed trajectories. Extensive experiments with real-life and synthetic datasets evaluate the different parts of TRACE, offering evidence that it is able to outperform the existing representative methods in terms of both compression ratio and transmission cost.

PVLDB Reference Format:
Tianyi Li, Lu Chen, Christian S. Jensen, Torben Bach Pedersen. TRACE: Real-time Compression of Streaming Trajectories in Road Networks. PVLDB, 14(7): 1175-1187, 2021.
doi:10.14778/3450980.3450987

1 INTRODUCTION

Massive volumes of vehicle trajectories are being accumulated at an unprecedented scale with the proliferation of GPS-enabled devices and mobile internet connectivity. This yields high storage and transmission costs for trajectories. Hence, trajectory compression that addresses these aspects has attracted attention [3–5, 8, 11, 13, 16, 17, 19, 27, 31–33, 41, 44, 45]. However, most existing studies target offline compression [11, 13, 16, 17, 19, 32, 33, 41, 44, 45]. They generally compress an entire trajectory after all the GPS points are collected, which may not be realistic for resource-constrained GPS-enabled devices. In particular, offline compression incurs high communication overheads or data loss because the data needs to be transmitted to the location where the compression is performed. In contrast, with online compression, GPS points are compressed as they arrive in real-time, thus enabling a broader range of applications, while saving both storage and transmission costs [3, 4, 21, 27].

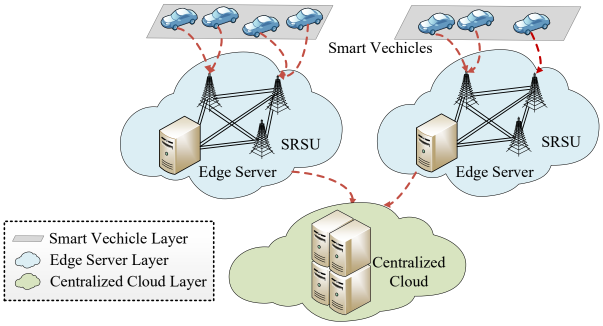


Figure 1: Vehicular edge computing architecture.

To enable compression on diverse devices with variable computing capabilities, we employ vehicular edge computing (VEC) to compress trajectories in real-time [3]. A VEC architecture has three layers: a smart vehicle layer, an edge server layer, and a centralized cloud layer [29], as shown in Figure 1. The smart vehicle layer delivers raw GPS data to the edge server layer that encompasses software-defined networking (SDN) based roadside units (SRSUs), which possess the computational and storage capabilities needed for trajectory compression. The centralized cloud layer collects and stores the compressed trajectories from the edge server layer to provide multiple services.

Although several studies [3–5, 31] consider online network-constrained trajectory compression, two challenges remain to be tackled. *The first challenge is how to obtain a concise and accurate representation of trajectories.* Existing studies obtain a compact representations in part by discarding information [3–5, 31], e.g., the exact locations of trajectories. This renders the resulting trajectories inaccurate and reduces their usability. *The second challenge is how to compress trajectories in real-time with low transmission costs.* Some previous proposals for streaming trajectory compression rely on offline training of prediction models using historical data, enabling them to omit data that can be predicted within a certain error bound [5, 31]. However, movement patterns on even the same road vary across time [12, 24], necessitating frequent re-training and incurring high transmission cost for delivering re-trained models.

To address the above two challenges, we propose a new framework for online TRAjectory ComprEssion (TRACE). The goals of

Proceedings of the VLDB Endowment, Vol. 14, No. 7 ISSN 2150-8097.
doi:10.14778/3450980.3450987

TRACE are to achieve high compression ratios and low transmission costs with acceptable time delays. To realize these, we first present a speed-based trajectory representation on the basis of UTCQ [19]. This representation removes redundant information while enabling decompression of trajectories by capturing growth rates of accumulative distances and vehicle speeds. Further, as existing studies indicate that subtrajectories from different streaming trajectories are likely to exhibit co-movement patterns during the same time periods [12], we make it possible to exploit the similarity between subtrajectories from different streaming trajectories by means of so-called referential compression. Next, we develop an effective online referential representation and a reference selection technique based on so-called k-mer matching, which employs hashing to identify matching subsequences [25]. To keep memory consumption low, we design a reference deletion algorithm that removes references that have not been used for some time. To be able to adapt to variable movement patterns, we present a reference rewriting algorithm that updates the references in real-time. Further, we provide a real-time data transmission scheme that targets low-overhead transmission of trajectory data. Finally, we develop an index structure and filtering techniques that facilitate real-time range querying of compressed trajectories.

In summary, our main contributions are as follows:

- We propose a new real-time streaming vehicle trajectory compression, transmission, and querying framework. To the best of our knowledge, this is the first such framework that does not depend on offline training and discard any data.
- We develop a concise speed-based representation and a k-mer matching based referential representation that use multiple references to capture the similarities between subtrajectories. A reference selection technique and reference deletion and rewriting functions are provided that further improve compression performance.
- We provide an effective data transmission scheme that reduces transmission overhead and supports decoding at the centralized cloud. We also propose an index structure and filtering techniques to accelerate real-time query processing.
- Extensive experiments offer insight into the workings of the different parts of the framework and show that it is able to outperform three baselines in terms of compression ratio and transmission cost.

The rest of the paper is organized as follows. We present preliminaries in Section 2 and give an overview of the proposed framework in Section 3. Section 4 details the representation. Section 5 presents the encoding and transmission schemes, and Section 6 covers the index structure and query processing. Section 7 reports the experimental results. Section 8 reviews related work, and Section 9 concludes and offers directions for future work.

2 PRELIMINARIES

We proceed to introduce preliminary definitions and algorithms. Table 1 summarizes frequently used notation.

Table 1: Frequently used notation.

Notation	Description
Tr	a set of streaming trajectories
Tr^n	a streaming trajectory in **Tr**
l_i	the i^{th} mapped GPS point
$sp(Tr^n)$	the path traversed by Tr^n
$ad(Tr^n)$	the accumulative distance sequence of Tr^n
$t(Tr^n)$	the time sequence of Tr^n
$SV(Tr^n)$	the start vertex of Tr^n
$E(Tr^n)$	the outgoing edge number sequence of Tr^n
$RD(Tr^n)$	the first relative distance of Tr^n
$GD(Tr^n)$	the growth rates of accumulative distances of Tr^n
$V(Tr^n)$	the speed sequence of Tr^n
$E(Tr^n)[i]$	the i^{th} outgoing edge number of $E(Tr^n)$
$E(Tr^n_i)$	the outgoing edge numbers arriving at $t(Tr^n)[i]$
Ref	a reference streaming trajectory
$Nref$	a non-reference streaming trajectory
$Com_\phi(Nref)$	the referential representation of $Nref$
G_o	the reference set at timestamp t_o
$G_o[i].f$	the freshness of the i^{th} reference in G_o
$G_o[i].tl$	the latest visiting timestamp of the i^{th} reference in G_o
F_o	the sum of freshness of references in G_o
FA	a factor matrix
$s\hat{e}q$	the binary code of a sequence seq

2.1 Data Model

A **raw trajectory** is a series of **raw GPS points** $p = ((x, y), t)$, where x is longitude, y is latitude, and t is a timestamp. $Tp^1 = \langle p_0, \cdots, p_7 \rangle$ in Figure 2a is an example of a raw trajectory. A **road network** is modeled as a directed spatial graph $G = (V, E)$, where V is a set of geo-located vertices v denoting intersections or end points, and E is a set of directed edges $e = (v_i \rightarrow v_j)$. Figure 2 gives a road network example. A **mapped GPS point** l is a network-constrained point in a road network G obtained by map-matching [34]. It is represented as $((v_i \rightarrow v_j), nd(v_i, l), t)$, where $nd(v_i, l)$ is the network distance between v_i and l on the edge $(v_i \rightarrow v_j)$ and t is a timestamp. In Figure 2a, $l_0 = ((v_0 \rightarrow v_1), 50,$ 7:03:25) is a mapped GPS point. We also denote a mapped GPS point as $((v_i \rightarrow v_j), nd(v_i, l))$ when the timestamp t is not considered.

Definition 1. *Given two vertices v_s and v_e in a road network G, a **path** sp is a sequence of connected edges $(v_i \rightarrow v_j)$ that starts from v_s and ends at v_e, i.e., $sp = \langle (v_s \rightarrow v_0), \cdots, (v_{n-1} \rightarrow v_e) \rangle$.*

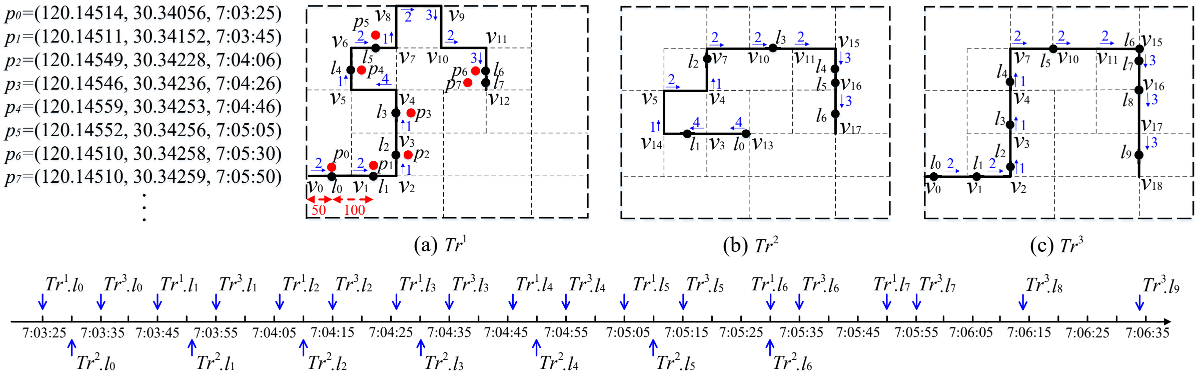
Definition 2. *A **streaming network-constrained trajectory** Tr^n is modeled as an infinite, time-ordered sequence of mapped GPS points L^n with an infinite path $sp(Tr^n)$ traversed by Tr^n.*

Figure 2 gives an example of a set $\mathbf{Tr} = \{Tr^1, Tr^2, Tr^3\}$ of three streaming network-constrained trajectories, where, e.g., $sp(Tr^1) = \langle (v_0 \rightarrow v_1), \cdots, (v_{11} \rightarrow v_{12}), \cdots \rangle$ and $L^1 = \{l_0, l_1, \cdots, l_7, \cdots\}$. **Tr** uses two road-network distances as defined next.

Definition 3. *The **accumulative distance** of a streaming trajectory Tr^n at its i^{th} timestamp $t(Tr^n)[i]$, denoted as $ad(Tr^n)[i]$, is the network distance $nd(v_s, l_i)$ along the path $(v_s \rightarrow v_e), \cdots, (v_{s^*} \rightarrow v_{e^*})$, where l_i is located on $(v_{s^*} \rightarrow v_{e^*})$ and $(v_s \rightarrow v_e)$ is the first edge traversed by Tr^n. The accumulative distance sequence $ad(Tr^n)$ of a streaming trajectory Tr^n contains the trajectory's accumulative distance at each timestamp.*

Definition 4. *Given a mapped GPS point $((v_s \rightarrow v_e), nd(v_s, l))$, the **relative distance** rd of l w.r.t. $(v_s \rightarrow v_e)$ is the ratio of $nd(v_s, l)$ to the length of $(v_s \rightarrow v_e)$ (denoted as $|(v_s \rightarrow v_e)|$).*

In Figure 2a, given $nd(v_0, l_1) = 150$, we have $ad(Tr^1)[1] = 150$. Given $|(v_0 \rightarrow v_1)| = 100$, rd of l_0 w.r.t. $(v_0 \rightarrow v_1)$ is 0.5. In the



p0=(120.14514, 30.34056, 7:03:25)
p1=(120.14511, 30.34152, 7:03:45)
p2=(120.14549, 30.34228, 7:04:06)
p3=(120.14546, 30.34236, 7:04:26)
p4=(120.14559, 30.34253, 7:04:46)
p5=(120.14552, 30.34256, 7:05:05)
p6=(120.14510, 30.34258, 7:05:30)
p7=(120.14510, 30.34259, 7:05:50)
⋮
50
100
Tr1.l0 Tr3.l0 Tr1.l1 Tr3.l1 Tr1.l2 Tr3.l2 Tr1.l3 Tr3.l3 Tr1.l4 Tr3.l4 Tr1.l5 Tr3.l5 Tr1.l6 Tr3.l6 Tr1.l7 Tr3.l7 Tr3.l8 Tr3.l9
7:03:25 7:03:35 7:03:45 7:03:55 7:04:05 7:04:15 7:04:25 7:04:35 7:04:45 7:04:55 7:05:05 7:05:15 7:05:25 7:05:35 7:05:45 7:05:55 7:06:05 7:06:15 7:06:25 7:06:35
Tr2.l0 Tr2.l1 Tr2.l2 Tr2.l3 Tr2.l4 Tr2.l5 Tr2.l6

(a) Tr^1

(b) Tr^2

(c) Tr^3

(d) Time line of Tr^1, Tr^2 and Tr^3

Figure 2: A streaming network-constrained trajectory set Tr = {Tr^1, Tr^2 and Tr^3}.

rest of the paper, we simply use “trajectory” instead of “network-constrained trajectory” when this does not cause ambiguity.

2.2 UTCQ Representation

Trajectory representation transforms network-constrained trajectories into a format with small entropy to achieve a high compression ratio [11, 13, 16, 19, 32, 41]. The UTCQ representation [19] is designed for compressing uncertain trajectories. Due to the uncertainty, a raw trajectory can be transformed to multiple trajectory instances by probabilistic map-matching [2]. UTCQ first adapts the representative trajectory representation TED [41] to express a trajectory instance Tr^n as a start vertex $SV(Tr^n)$, an edge sequence $E(Tr)$, a relative distance sequence $D(Tr)$, a time flag bit-string $T'(Tr)$, and a timestamp sequence $T(Tr)$.

Example 1. *Assuming that the default sample interval of* **Tr** *in Figure 2 is 20s, the UTCQ representation of Tr^1 is i) $SV(Tr^1) = 44183$; ii) $E(Tr^1) = \langle 2, 2, 1, 1, 4, 1, 2, 1, 2, 3, 2, 3, 0\rangle$; iii) $D(Tr^1) = \langle 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.75\rangle$; iv) $T'(Tr^1) = \langle 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1\rangle$ and v) $T(Tr^1) = \langle 7{:}03{:}25, 0, 1, 0, 0, -1, 5, 0\rangle$. The single 0 and the last 3 (3 is the outgoing edge number of $(v_{11} \to v_{12})$ w.r.t. v_{11}) in $E(Tr^1)$ indicate that $(v_{11} \to v_{12})$ has two mapped GPS points, l_6 and l_7. $T'(Tr)$ is introduced to map relative distances in $D(Tr)$ to outgoing edge numbers in $E(Tr)$ for decompression [41].*

Next, UTCQ exploits the similarity between trajectory instances of a single uncertain trajectory and provides **referential representations** for the edge sequence $E(Tr)$, the distance sequence $D(Tr)$, and the time flag bit-string $T'(Tr)$. It encodes the differences of an input sequence w.r.t. a reference sequence. (i.e., the more similar the sequences are, the higher the compression ratio) [9, 38, 39].

Definition 5. *Given an input sequence $\phi(Nref)$ (also called a non-reference) and its corresponding reference $\phi(Ref)$, $\phi(Nref)$ can be represented as a list of W* **factors***, i.e., $Com_\phi(Nref) = \langle \phi(Ma_w) | 0 \leq w < W\rangle$, where a factor $\phi(Ma_w)$ denotes a subsequence in $\phi(Nref)$.*

Here, ϕ identifies the to-be-represented sequence of a trajectory. For example, $Com_E(Nref)$ is the referential representation of the edge sequence $E(Nref)$. UTCQ adopts the (S, L, M) format to encode each factor in $Com_\phi(Nref)$. Specifically, S is the *start position* of the subsequence in the reference, L is the *length* of the subsequence, and M is the first *mis-matched character* following the subsequence. For example, in Figure 2, we have $E(Tr^2) = \langle 4, 4, 1, 2, 1, 2, 2, 2, 3, 0, 3\rangle$ and $E(Tr^3) = \langle 2, 2, 1, 1, 1, 2, 2, 2, 3, 0, 3, 3\rangle$. Let Tr^3 be a non-reference *Nref* and Tr^2 be its corresponding reference *Ref*. We get $Com_E(Nref) = \langle(5, 2, 1), (2, 1, 1), (5, 6, 3)\rangle$. Note that UTCQ only assigns one reference to each non-reference.

2.3 *k*-mer Matching

Developed for genome sequences, k-mer matching is an effective strategy for high-speed referential compression [25].

Definition 6. *A* ***k*-mer** *ϕ_i^n is a subsequence of fixed length k of $\phi(Tr^n)$, i.e., $\phi_i^n = \{\phi(Tr^n)[i], \phi(Tr^n)[i+1], ..., \phi(Tr^n[i+k-1])\}$.*

Given a reference $\phi(Ref)$ and a non-reference $\phi(Nref)$, k-mer matching employs a hash table H, to efficiently obtain the referential representation $Com_\phi(Nref)$. Each factor of $Com_\phi(Nref)$ is of the form (S, L, M), where L is the length of the matched subsequence. We highlight that k is different from L; thus, k is kept fixed during compression. To be specific, for a reference $\phi(Ref)$, k-mer matching first computes the hash key of each k-mer in $\phi(Ref)$ by a given hash function. Next, each k-mer is stored with its start position S in $\phi(Ref)$ in H. For a non-reference $\phi(Nref)$, k-mer matching greedily finds the longest prefix of $\phi(Nref)$ that exists in $\phi(Ref)$ with the help of H. In particular, it calculates the hash key of each k-mer in $\phi(Nref)$ and checks whether a matched subsequence exists in H. If it exists, it continues to match the subsequent characters in $\phi(Nref)$ and $\phi(Ref)$ until an un-matched character M occurs. The procedure implies that L can be an **arbitrary value** with $L \geq k$, meaning that k can remain fixed during compression. It also means that the correctness of k-mer matching is un-affected by k.

Example 2. *Consider $E(Tr^1)$ as a reference and $E(Tr^3)$ as a non-reference. Assuming that the current timestamp is 7:04:55, we have four k-mers for $E(Tr^1)$, i.e., $E_0^1 = (2, 2, 1)$, $E_1^1 = (2, 1, 1)$, $E_2^1 = (1, 1, 4)$, and $E_3^1 = (1, 4, 1)$, each of which is mapped to a hash table with its corresponding start position in $E(Tr^1)$. To find the longest prefix of $E(Tr^3)$ in $E(Tr^1)$, we get the first k-mer of $E(Tr^3)$, i.e., $E_0^3 = \langle 2, 1, 1\rangle$, and finds $E_0^1 = E_0^3$ in the hash table; then we greedily match the subsequent characters of both E_0^3 and E_0^1 until the first mis-matched character (i.e., $E(Tr^3)[4] = 1$) occurs. Thus, we obtain a factor $(0, 4, 1)$, where 0 is the offset of $E_0^1[0]$ in $E(Tr^1)$. After that, we remove the subsequence represented by $(0, 4, 1)$ from $E(Tr^3)$. Since $E(Tr^3) = \emptyset$, the k-mer matching stops.*

Note that k is a **pre-defined** and **fixed** value that only constrains the initial length of a matched subsequence. We adapt k-mer

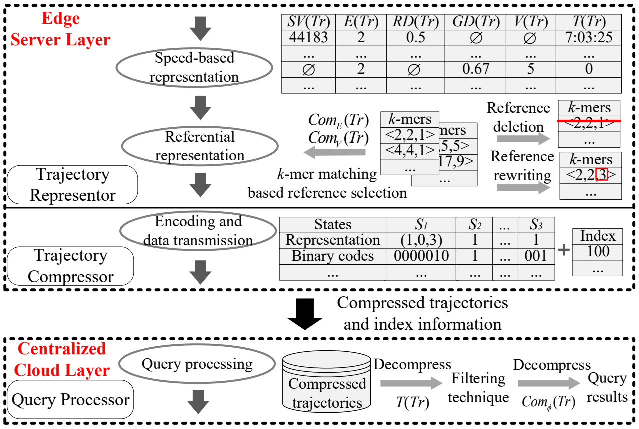


Figure 3: TRACE framework.

matching to streaming settings and detail how to select references and referentially compress trajectories in real-time.

3 TRACE FRAMEWORK

The framework enables the online compression and subsequent querying of streaming network-constrained trajectories (NCTs). It takes a set $\mathbf{Tr} = \{Tr^n | 1 \leq n \leq N\}$ of streaming trajectories as input. Each Tr^n contains i) the streaming path $sp(Tr^n)$ traversed by Tr^n, ii) a streaming sequence of accumulative distances $ad(Tr^n)$, and iii) a streaming time sequence $t(Tr^n)$. Figure 3 depicts the TRACE framework that encompasses three components: a trajectory representor, a trajectory compressor, and a query processor. We deploy TRACE at the edge server and the centralized cloud layers of VEC. Raw GPS points are delivered from the smart vehicle layer to the edge server layer, where they are transformed to mapped GPS points using map-matching [34]. We do this to support smart vehicle layers with limited computational capabilities [4], thus extending the applicability. While the cost of data transmission from the smart vehicle layer to the edge server layer equals the size of the raw trajectories, this transmission is relatively efficient and scalable, because the edge server layer is geographically closer to the smart vehicle layer than to the centralized cloud layer, reducing the wireless communication energy consumption and the network load [20, 29].

Edge Server Layer. The ***trajectory representor*** converts the NCTs into speed-based and referential representations. In particular, we propose to use speed information to identify the mapped GPS point to achieve a compact representation format. Then, we apply the referential representation in [19] to exploit the similarities among subtrajectories and multiple references to achieve a high compression ratio, with the details presented in Sections 4.1 and 4.2. To select references and to represent non-references in online scenarios, we adopt k-mer matching [25]. Specifically, two hash tables are used to store the reference sets $E(\mathbf{Ref})$ and $V(\mathbf{Ref})$ for fast retrieval. Reference deletion and rewriting functions are provided that make it possible to update the tables efficiently. The former enables deleting infrequent sequences to reduce the storage cost, while the latter updates references using frequent sequences to improve the compression ratio. The detailed algorithms are covered in Sections 4.3–4.5. In the sequel, the ***trajectory compressor*** compresses the references and non-references into binary codes. A scheme for transmitting binary codes is also presented, which enables the centralized cloud to decode binary codes without extra information. The details are provided in Section 5.

Centralized Cloud Layer. The ***query processor*** is located at the centralized cloud and operates on compressed trajectories. It has an online range query algorithm that exploits indexing and filtering to achieve efficiency, to be detailed in Section 6.

4 REPRESENTATION

We proceed to detail the representation of streaming trajectories.

4.1 Speed-based Representation

Existing works show that objects moving on the same road during the same time period tend to have a similar speed trend [12]. This motivates us to transfer the accumulative distances of trajectories to speeds and apply the referential representation to them. However, the reference speed sequence is difficult to compress substantially due to its wide range [41]. We tackle this challenge by developing the following representation.

$\boldsymbol{ad(Tr) \rightarrow (RD(Tr), GD(Tr), V(Tr))}$. We represent accumulative distances as a sequence of growth rates (denoted as $GD(Tr)$) following the first relative distance of Tr (denoted as $RD(Tr)$). The growth rate of the accumulative distance at $T(Tr)[i]$, denoted as $GD(Tr)[i]$ ($i > 0$), is calculated by $\frac{ad(Tr)[i]-ad(Tr)[i-1]}{ad(Tr)[i]-ad(Tr)[i-2]}$ ($i > 1$). Further, $GD(Tr)[1] = \frac{ad(Tr)[1]-ad(Tr)[0]}{ad(Tr)[1]}$. For example in Figure 2a, given $ad(Tr^1)[0] = 50$, $ad(Tr^1)[1] = 150$ and $ad(Tr^1)[2] = 250$, we have $GD(Tr^1)[1] = 0.67$ and $GD(Tr^1)[2] = 0.5$. $V(Tr)$ is a sequence of speeds. The speed at $t(Tr)[i]$ is calculated by $\frac{ad(Tr)[i]-ad(Tr)[i-1]}{t(Tr)[i]-t(Tr)[i-1]}$ ($i > 0$), denoted as $V(Tr)[i]$. Given $T(Tr)$, both $GD(Tr)$ and $V(Tr)$ are able to identify $ad(Tr)$. Thus, we only store one of them, to be detailed in Section 4.2.

$\boldsymbol{sp(Tr) \rightarrow (SV(Tr), E(Tr))\ \&\ t(Tr) \rightarrow T(Tr)}$. We adopt UTCQ [19] to represent a path $sp(Tr)$ as $SV(Tr)$ followed by $E(Tr)$ and a time sequence $t(Tr)$ as $T(Tr)$. We use the UTCQ representation because it is the state-of-the-art referential compression framework for network-constrained trajectories and exhibits high compression ratios. Note that, since both $GD(Tr)$ and $V(Tr)$ enable decompressing trajectories without $T'(Tr)$ and the "0" in $E(Tr)$ used in UTCQ representation, we omit them to achieve a more compact format.

Overall, the speed-based representation expresses Tr as a tuple $(SV(Tr), E(Tr), RD(Tr), GD(Tr), V(Tr), T(Tr))$. Table 2 shows an example representation for $\mathbf{Tr}$ in Figure 2.

4.2 Representation with Multiple-References

Based on the representation introduced in Section 4.1, we apply referential compression [19] to sub-trajectories from different streaming trajectories. Due to the use of multiple references, we modify the (S, L, M) format [19] as (ref_{id}, S, L, M), where ref_{id} is the ID of the reference. For example, the referential representation of $E(Tr^3)$ w.r.t. $E(Tr^1)$ and $E(Tr^2)$ is $Com_E(Tr^3) = \langle(1, 0, 4, 1), (2, 5, 5, 3)\rangle$.

Different from the outgoing edge numbers, the speeds are unlikely to be exactly the same. Therefore, we consider $V(Tr^n)[i] \approx V(Tr^{n'})[i']$ if $\ddot{V}(Tr^n)[i] = \ddot{V}(Tr^{n'})[i']$, where $\ddot{V}(Tr^n)[i]$ is the integer closest to $\frac{V(Tr^n)[i]}{0.5\eta}$ and η is the speed error bound. A larger η yields a more accurate compression at the expense of the compression ratio. If $V(Tr)$ is a non-reference, we record $GD(Tr)[i]$ if

Table 2: Speed-based representation of Tr in Figure 2.

n	1	2	3
$SV(Tr^n)$	44183	27444	44183
$E(Tr^n)$	$\langle 2, 2, 1, 1, 4, 1, 2, 1, 2, 3, 2, 3 \rangle$	$\langle 4, 4, 1, 2, 1, 2, 2, 2, 3, 3 \rangle$	$\langle 2, 2, 1, 1, 1, 2, 2, 2, 3, 3, 3 \rangle$
$RD(Tr^n)$	0.5	0.05	0.25
$GD(Tr^n)$	$\langle 0.67, 0.5, 0.5, 0.67, 0.33, 0.83, 0.05 \rangle$	$\langle 0.97, 0.69, 0.35, 0.53, 0.11, 0.75 \rangle$	$\langle 0.8, 0.5, 0.5, 0.5, 0.64, 0.53, 0.11, 0.75, 0.67 \rangle$
$V(Tr^n)$	$\langle 5, 4.76, 5, 10, 5.26, 20, 1.25 \rangle$	$\langle 6.9, 17.11, 8.75, 10, 1.25, 3.75 \rangle$	$\langle 5, 5, 5, 5, 8.75, 10, 1.25, 3.95, 7.5 \rangle$
$T(Tr^n)$	⟨7:03:25, 0, 1, 0, 0,-1, 5, 0⟩	⟨7:03:30, 1,-1, 0, 0, 0, 0⟩	⟨7:03:35, 0, 0, 0, 0, 0, 0, 0,-1, 0⟩

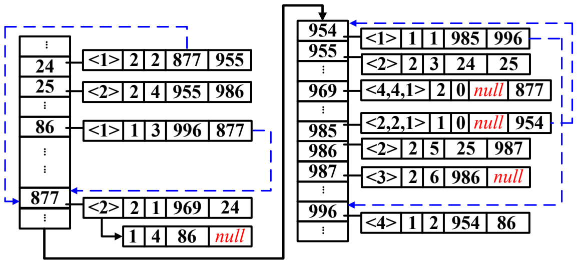


Figure 4: A hash table *H* constructed according to *E*(Tr) in Table 2 at 7:05:06, where *k=3*.

$V(Tr)[i]$ is a mis-matched value w.r.t. its reference; otherwise, we store $GD(Tr)$ instead of $V(Tr)$. This is because $V(Tr)[i]$ ($\in [0, \mu]$) can only achieve the same compression as $GD(Tr)[i]$ ($\in [0, 1)$) at the cost of compression accuracy, where μ is a speed constraint of the road network. Taking $V(Tr^1)$ and $V(Tr^2)$ as references and given $\eta = 0$, we have $Com_V(Tr^3) = \langle (1, 0, 3, 0.5), (2, 2, 4, 0.67) \rangle$.

4.3 Reference Selection for $E(Tr^n)$

We introduce the real-time reference selection technique based on k-mer matching. Note that, we only decompose $E(Tr^n)$ and $V(Tr^n)$ into k-mers, i.e., $\phi = E$ or V, as we only apply referential representation to them.

Definition 7. *The subsequence of $E(Tr)$ of a streaming trajectory Tr arriving at $t(Tr)[j]$ ($j > 0$), is denoted as $E(Tr_j)$ ($j > 0$). It starts from the edge traversed by Tr immediately after leaving the last edge of $E(Tr_{j-1})$ and ends at the edge where Tr is located at $t(Tr)[j]$.*

Specifically, $E(Tr_0)$ is the outgoing edge number of $(v_s \to v_e)$ w.r.t. v_s, where $(v_s \to v_e)$ is the first edge traversed by *Tr*. For example, in Figure 2a, we have $E(Tr^1_4) = \langle 4, 1 \rangle$ because $(v_4 \to v_5)$ is traversed by Tr^1 after leaving $(v_3 \to v_4)$ and l_4 is located on $(v_5 \to v_6)$ at $T(Tr^1)[4]$.

We construct a hash table H for $E(Tr^n)$, whose cardinality (i.e., number of hash entries) is pre-defined and fixed. Each k-mers in H is stored as a tuple $(E^n_i, n, off_i, pt_i, pd_i)$, where i) E^n_i is the i^{th} k-mer of $E(Tr^n)$; ii) n is the ID of Tr^n; iii) off_i is the offset of $E^n_i[0]$ in $E(Tr^n)$, and iv) pt_i and pd_i are the indexes of the entries associated with E^n_{i-1} and E^n_{i+1} in H, respectively. Figure 4 gives an example of the hash table constructed according to the streaming trajectories in Table 2, where BKDR hashing is adopted due to its high efficiency and good distribution capability [40]. To save space, we only store $E^n_i[k-1]$ ($i > 0$) instead of E^n_i in the hash table because the whole k-mer can be retrieved by pt_i. For example, the entry associated with E^1_1 ($= \langle 2, 1, 1 \rangle$) in H is stored as $(\langle 1 \rangle, 1, 1, 985, 996)$, where 985 and 996 are the indexes of E^1_0 and E^1_2 of Tr^1 in H, respectively.

A k-mer does not form until $|E(Tr^n_m) \cup E(Tr^n_{m+1}) \cup \cdots \cup E(Tr^n_{m+j})| \geq k$. For example given $k = 3$, E^1_0 is formed at $t(Tr)[2]$ =7:04:06, as $E(Tr^1_0) \cup E(Tr^1_1) \cup E(Tr^1_2) = \langle 2, 2, 1 \rangle$. Once the first k-mer E^n_0 of $E(Tr^n)$ is formed, we hash it to hash table H according to its hash key, denoted as *key*.

1) If $H[key] = \emptyset$, E^n_0 cannot be referentially represented by the existing k-mers in H, as it is distinct from all of them. Hence, we mark $E(Tr^n)$ as a reference and create a tuple for E^n_0 as well as all the subsequent k-mers E^n_i ($i > 0$), in order to prepare for referentially representing other sequences.

2) If $H[key] \neq \emptyset$ and a tuple $(E^{n'}_{i'}, n', off_{i'}, pt_{i'}, pd_{i'})$ exists with $E^{n'}_{i'} = E^n_0$, Tr^n is marked as a non-reference. Then, we initialize a factor $E(Ma_0) = (n', off_{i'}, k, \emptyset)$ for $E(Tr^n)$, where k is the current matched length and $\emptyset$ indicates that the mis-matched value of $E(Ma_0)$ is unknown. Next, we follow an existing work [25] to greedily match the subsequent character $E^n_{i+1}[k-1]$ ($i > 0$) with that in $E(Tr^{n'})$. This process ends when a mis-matched character M occurs. So far, a factor (n', S, L, M) is generated, where L is the final matched length of the subsequence of $E(Tr^{n'})$ and $S = off_{i'}$. Then we wait until another k-mer forms and repeat the process. Clearly, the time delay occurs mainly when waiting for k mapped GPS points before initializing a factor.

Example 3. *Following the example shown in Figure 4 and Table 2, since E^3_0 matches with E^1_0, we initialize a factor $(1, 0, 3, \emptyset)$ for it. For the arriving $E(Tr^3_3) = \langle 1 \rangle$, we retrieve $E(Tr^1)[3]$ according to the index pd_0 associated with E^1_0, i.e, 954, and compare it with $E(Tr^3_3)$ ($= E^3_1[2]$). After that, we update the factor to $(1, 0, 4, \emptyset)$ due to $E(Tr^1)[3] = E(Tr^3_3)$. As $E(Tr^1)[4] \neq E^3_2[2]$, we generate a factor, i.e., $(1, 0, 4, 1)$, for $Com_E(Tr^3)$. Then, we wait until $E(Tr^3_5)$ arrives due to $|E(Tr^3_5) \cup E(Tr^3_6)| \geq 3$ and repeat the process. Finally, we get $Com_E(Tr^3) = \langle (1, 0, 4, 1), (2, 5, 5, 3) \rangle$.*

However, a streaming trajectory Tr^n can be a non-reference at the beginning but cannot be referentially represented since a timestamp t_i, because its k-mer formed at t_i cannot match any tuple stored in the hash table H. In this case, we call Tr^n as a *hy-reference* and create a tuple for each k-mer of Tr^n generated since t_i to referentially represent other sequences. Intuitively, the former part of a hy-reference Tr^n is a non-reference that is referentially represented, while the latter part of it is a reference.

We store a repetitive E^n_i as (n, off_i, pt_i, pd_i) by omitting the redundant E^n_i. We associate this tuple with $(E^{n'}_{i'}, n', off_{i'}, pt_{i'}, pd_{i'})$ ($E^{n'}_{i'} = E^n_i$) (i.e., they are stored in the same hash entry) to trace back the subsequence of Tr^n during k-mer matching. As shown in Figure 4, the tuple of E^1_4, i.e., $(1, 4, 86, \textit{null})$ is associated with that of E^2_1, as $E^1_4 = E^2_1$. Here, *null* indicates that E^1_5 is unknown at the current timestamp. Moreover, E^n_i and $E^{n'}_{i'}$ may be stored in the same hash entry even if $E^n_i \neq E^{n'}_{i'}$ due to hash collisions. According the introduction of k-mer matching in Section 2.3, k remains unchanged during compression and only determines the length of the initial matched subsequence. However, k cannot be too large or too small. A too large k may result in a very high probability of failed matching and thus excessively many references, and a too small k may lead to many "trivial" matches, where each factor of a non-reference represents a very short subsequence. We study the impact of k on compression performance in Section 7.

4.4 Reference Deletion for $E(Tr^n)$

The number of k-mers stored in the hash table increases over time. To reduce the storage cost, we delete the k-mers corresponding to references from the hash table that have not been visited for a long time. A reference $E(Ref)$ is **visited** if i) it referentially represents a non-reference or ii) its corresponding data is still arriving. We define G_o as the set of the references at timestamp t_o. Specifically, $G_o[i]$, that represents a reference $E(Ref)$, denotes the tuple $(ref_{id}, G_o[i].tl)$, where ref_{id} is the ID of the reference $G_o[i]$, and $G_o[i].tl$ is the timestamp when $G_o[i]$ was most recently visited.

Definition 8. *A reference $G_o[i]$ is* ***outdated*** *at t_o if its* ***freshness*** *at t_o, denoted as $G_o[i].f$, satisfies $G_o[i].f < C \cdot \frac{F_o}{|G_o|}$, where $F_o = \sum_{i'=0}^{|G_o|-1} G_o[i'].f$ and C $(0 < C \leq 1)$ is the deletion coefficient.*

We set the deletion coefficient $C \in (0, 1]$ because we consider a reference as outdated only if its freshness is below the average freshness $\frac{F_o}{|G_o|}$. A larger C implies that references expire more easily. We calculate $G_o[i].f$ as follows:

$$G_o[i].f = \lambda^{t_o - G_o[i].tl} \quad (t_o \geq G_o[i].tl), \tag{1}$$

where $\lambda \in (0, 1)$ is a decay factor [7]. In Figure 2, given the current timestamp t_o =7:03:26 and $\lambda = 0.998$, we get $E(Tr^1).f = 0.998$ at t_o.

Definition 8 is more effective than using a fixed threshold to determine whether a reference is expired. Specifically, using a fixed threshold, if no trajectory arrives for a long time due to an occasional interrupt of communications, all references in the hash table will expire. A naive solution is to update the freshness of each reference in the hash table at each timestamp in order to compute the average freshness and then to identify the outdated data. To improve efficiency, we propose to compute the freshness of parts of $G_o[i]$ $(0 \leq i < |G_o|)$ at t_o. We sort G_o in ascend order of $G_o[i].f$ $(0 \leq i < |G_o|)$. Then, we detect expired data in order until $G_o[i'].f \geq C \cdot \frac{F_o}{|G_o|}$ $(0 < i' < |G_o|)$. Specifically, no k-mers are deleted if $G_o[0].f \geq C \cdot \frac{F_o}{|G_o|}$. This process can actually be realized without sorting, as shown in Algorithm 1. Next, we define a set containing the references in G_o that are visited at t_o, denoted as Rv_o $(Rv_o \subseteq G_o)$. Given $F_{o'}$, F_o $(t_o > t_{o'})$ is updated as follows:

$$F_o = (F_{o'} - \sum_{G_{o'}[i] \in Rv_o} G_{o'}[i].f) \cdot \lambda^{t_o - t_{o'}} + |Rv_o| \tag{2}$$

Formula 2 enables us to update F_o by only computing the freshness of $G_{o'}[i]$, such that $G_{o'}[i] \in Rv_o$. The derivation of it is omitted due to the space limitation.

Algorithm 1 details the reference deletion. Assuming that the latest timestamp when new data arrives is $t_{o'}$. The algorithm searches each $G_{o'}[i] \in Rv_o$ in the current reference set $G_{o'}$ and removes $G_{o'}[i]$ from $G_{o'}$ if it exists in Rv_o (Lines 1–4). Then, each reference $E(Ref) \in Rv_o$ is appended to $G_{o'}$ (Line 6). This way, it sorts $G_{o'}$ in ascending order of freshness. Meanwhile, Formula 2 is applied to calculate F_o (Lines 5 and 7) and $G_{o'}$ is updated to G_o. After "sorting", we check the freshness of references in G_o by starting from its first reference every time, in order to only calculate the freshness of the potentially outdated references (Lines 8–11). Finally, the updated references G_o and the sum of their freshness F_o are returned (Line 12). This approach reduces the time complexity of updating freshness from $O(|G_o|)$ to $O(|Rv_o| + |EX_o|)$, where EX_o is the set of expired references at t_o. We evaluate the effectiveness and efficiency of reference deletion via experiments in Section 7.

Algorithm 1: Reference Deletion Algorithm

```
   Input: the reference set $G_{o'}$ at $t_{o'}$, the set $Rv_o$, the sum of
          freshness $F_{o'}$ and a threshold $C$
   Output: the reference set $G_o$ and the sum of freshness $F_o$
 1 for each reference $E(Ref) \in Rv_o$ do
 2     if $E(Ref) \in G_{o'}$ with $E(Ref) = G_{o'}[i]$ then
 3         compute $G_{o'}[i].f$ using Formula 1
 4         remove the tuple $(ref_{id}, G_{o'}[i].tl)$ from $G_{o'}$
 5         $F_{o'} \leftarrow F_{o'} - G_{o'}[i].f$
 6     append the tuple $(ref_{id}, t_o)$ to $G_{o'}$
 7 $F_o \leftarrow F_{o'} \cdot \lambda^{t_o - t_{o'}} + |Rv_o|$
 8 while $\lambda^{t_o - G_{o'}[0].tl} < C \cdot \frac{F_o}{|G_{o'}|}$ do
 9     $F_o \leftarrow F_o - \lambda^{t_o - G_{o'}[0].tl}$
10     remove the tuple $(ref_{id}, G_{o'}[0].tl)$ from $G_{o'}$
11     delete the k-mers associated with $G_{o'}[0]$ from the hash table
12 return $G_{o'} (= G_o)$ and $F_o$
```

4.5 Reference Rewriting for $E(Tr^n)$

To further improve the compression ratio, we rewrite the references $E(Ref)$ in real-time. The motivation is that a non-frequent edge sequence may become a reference if it arrives early. For example, given a frequent path $\langle 44183, 2, 2, 1, 1, 1, 2, 2 \rangle$, $\langle 2, 2, 1, 1, 1, 2, 2 \rangle$, i.e., a subsequence of $E(Tr^3)$, should be stored as k-mers. However, since Tr^1 arrives before Tr^3 and $E(Tr^1)$ can referentially represent $E(Tr^3)$, $E(Tr^3)$ is made as a non-reference and Tr^1 is a reference, as shown in Example 3. In this case, for another new arriving subsequent Tr^n that also traverses $\langle 44183, 2, 2, 1, 1, 1, 2, 2 \rangle$, $Com_E(Tr^n)$ will contain at least two factors. To address the problem, we detect the frequent subsequence and rewrite the k-mers in real-time. Following an existing study [39], we define the rewriting candidate below.

Definition 9. *A* ***rewriting candidate*** *is a reference $E(Ref)$ that represents a non-reference $E(Nref)$ as $Com_E(Nref) = \langle \cdots, (ref_{id}, S, L, M), (ref_{id}, S', L', M'), \cdots \rangle$, where $S + L + 1 = S'$ and ref_{id} is the ID of the reference Ref.*

Given a rewriting candidate *Ref*, we can **merge** two factors (ref_{id}, S, L, M) and (ref_{id}, S', L', M') into one factor $(ref_{id}, S, L + 1 + L', M')$, by replacing $E(Ref)[S+L]$ with M, which is called a rewriting operation.

Example 4. *Given $E(Tr^4) = \langle 2, 2, 1, 1, 4, 1, 3, 1, 2, 3, 3 \rangle$ such that Tr^4 arrives after Tr^1 ends, $E(Tr^1)$ becomes a rewriting candidate as $Com_E(Tr^4) = \langle (1, 0, 6, 3), (1, 7, 3, 3) \rangle$. If $E(Tr^1)[6]$ is replaced with 3, $Com_E(Tr^4)$ will only contain one factor, i.e., $(1, 0, 10, 3)$, which leads to a higher compression. Similarly, given $E(Tr^5) = \langle 1, 1, 4, 1, 3, 1, 2, 3, 3 \rangle$ such that Tr^5 also arrives after Tr^1 terminates, the compression can be improved by replacing $E(Tr^1)[6]$ with 3.*

Algorithm 2: Reference Rewriting Algorithm

```
Input: a rewriting operation, i.e., replacing $E(Ref)[i]$ with $M$, a
       $b \times b$ factor matrix FA of $E(Ref)$, an array $rp$ of $E(Ref)$ and
       a threshold $\alpha$
Output: a rewritten reference or $\emptyset$
1  $f(M) \leftarrow f(M) + 1$, where $f(M)$ is associated with $rp[i]$
2  if $\forall M' \in rp[i]\ (f(M) \geq f(M')) \wedge f(M) \geq \alpha$ then
3      if $\sum_{x=1}^{\lfloor \frac{i}{k} \rfloor+1} \sum_{y=\lfloor \frac{i}{k} \rfloor+1}^{b} FA^{xy} < f(M)$ then
4          replace $E(Ref)[i]$ with $M$
5          update $k$-mers associated with $E(Ref)$
6          delete FA and $rp$ of $E(Ref)$
7          return a rewritten reference
8      else
9          return $\emptyset$        /* no operation will be conducted */
10 else
11     return $\emptyset$        /* no operation will be conducted */
```

We construct an array rp for each $E(Ref)$, where $rp[i]$ is a list recording M corresponding to a rewriting operation for $E(Ref)[i]$ ($1 \leq i < |E(Ref) - 1|$) and its frequency of occurrence, denoted as $f(M)$. Following Example 4, $rp[6]$ of $E(Tr^1)$ is $\langle(3, 2)\rangle$, due to $M = 3$ and $f(M) = 2$. Obviously, a prerequisite for conducting a rewriting operation is that $f(M)$ should be large. This is to guarantee that a frequent subsequence (i.e., a frequent path) is generated by the rewriting. Moreover, we should ensure that the factors overlapping $E(Ref)[i]$ occur rarely. Given a factor (ref_{id}, S, L, M), it intersects $E(Ref)[i]$, if ref_{id} is the ID of Ref and $S \leq i < S + L$.

Example 5. *Given $E(Tr^6) = \langle 2, 2, 1, 1, 4, 1, 2, 1, 2, 2, 2, 2, 1, 3\rangle$ such that Tr^6 arrives after Tr^1 terminates, we get $Com_E(Tr^6) = \langle(1, 0, 9, 2), (1, 0, 3, 3)\rangle$, where $(1, 0, 9, 2)$ intersects $E(Tr^1)[6]$. Following Example 4, even if replacing $E(Tr^1)[6]$ with 3 reduces the number of factors of both $Com_E(Tr^4)$ and $Com_E(Tr^5)$, it produces one more factor for $Com_E(Tr^6)$. This indicates that we should rewrite $E(Tr^1)[6]$ only when it does not frequently intersect any factors.*

The above analysis implies that we should record each factor that intersects $E(Ref)[i]$ ($1 \leq i < |E(Ref)| - 1$) for rewriting. Intuitively, this results in a high space-time consumption. Inspired by the regular square grid graph [10], we construct a $b \times b$ factor matrix **FA** for each $E(Ref)$, where $b = \lceil \frac{|E(Ref)|_r}{k} \rceil$. Here, $|E(Ref)|_r$ is the length of the subsequence of $E(Ref)$ used as a reference. $FA[x][y]$ ($x, y > 0$), denoted as FA^{xy}, is associated with a subsequence of $E(Ref)$ in its factor matrix, i.e., $\langle E(Ref)[(x - 1) \cdot k], \cdots, E(Ref)[y \cdot k - 1]\rangle$, and counts the frequency of factors (ref_{id}, S, L, M) intersecting $E(Ref)[i]$ ($(x - 1) \cdot k \leq i \leq y \cdot k - 1$), where ref_{id} is the ID of Ref. Thus, we update **FA** once a factor is generated.

Proposition 4.1. *A factor (ref_{id}, S, L, M) contributes to FA^{xy}, where $x = \lfloor \frac{S}{k} \rfloor + 1$ and $y = \lceil \frac{S+L}{k} \rceil$.*

Example 6. *Following Examples 3, 4, and 5 and assuming that* $\mathbf{Tr} = \{Tr^1, Tr^2, Tr^3, Tr^4, Tr^5, Tr^6\}$*, the factor matrix* **FA** *of $E(Tr^1)$ is:*

$$\mathbf{FA} = \begin{bmatrix} [0,2] & [0,5] & [0,8] & [0,11] \\ & [3,5] & [3,8] & [3,11] \\ & & [6,8] & [6,11] \\ & & & [9,11] \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 3 & 1 & 0 \\ & 0 & 0 & 0 \\ & & 0 & 2 \\ & & & 0 \end{bmatrix}$$

Here, the left part of **FA** *intuitively gives the subsequences of $E(Tr^1)$ corresponding to FA^{xy}, and the right part shows the value of FA^{xy}. For instance, $FA^{12} = 3$ is contributed by three factors, i.e, $(1, 0, 4, 1)$, $(1, 0, 6, 3)$, and $(1, 2, 4, 3)$.*

Lemma 1. *Given a factor (ref_{id}, S, L, M) that intersects $E(Ref)[i]$ and a $b \times b$ factor matrix* **FA** *of $E(Ref)$, (ref_{id}, S, L, M) can only contribute to FA^{xy}, where $x \leq \frac{i}{k} + 1 \wedge y > \frac{i}{k}$.*

Proof. As (ref_{id}, S, L, M) intersects $E(Ref)[i]$, we have $S \leq i < S + L$. Proposition 4.1 guarantees that $S \geq (x - 1) \cdot k \wedge S + L \leq y \cdot k$ if (ref_{id}, S, L, M) contributes to FA^{xy}. Hence, we have $i \geq (x - 1) \cdot k \wedge i < y \cdot k$, i.e., $x \leq \frac{i}{k} + 1 \wedge y > \frac{i}{k}$. □

Lemma 2. *Given a $b \times b$ factor matrix* **FA** *of $E(Ref)$, the maximum number of factors that intersect $E(Ref)[i]$ is $\sum_{x=1}^{\lfloor \frac{i}{k} \rfloor+1} \sum_{y=\lfloor \frac{i}{k} \rfloor+1}^{b} FA^{xy}$.*

Proof. Following Lemma 1, the factors intersecting $E(Ref)[i]$ can only contribute to FA^{xy}, where $x \leq \frac{i}{k} + 1 \wedge y > \frac{i}{k}$. As a result, the maximum number of the factors that intersect $E(Ref)[i]$ is $\sum_{x=1}^{\lfloor \frac{i}{k} \rfloor+1} \sum_{y=\lfloor \frac{i}{k} \rfloor+1}^{b} FA^{xy}$. □

Example 7. *Continuing Example 6, the maximum number of the factors intersecting $E(Tr)[6]$ is $\sum_{x=1}^{3} \sum_{y=3}^{4} FA^{xy} = 3$, where only 6 out of 10 elements in FA need to be visited.*

Based on the above conclusions, we present the conditions for implementing a rewriting operation, i.e, replacing $E(Ref)[i]$ with M: i) $f(M) \geq \alpha$, where $\forall M' \in rp[i]$ $(f(M) \geq f(M'))$ and α (≥ 1) is the rewriting coefficient, and ii) $\sum_{x=1}^{\lfloor \frac{i}{k} \rfloor+1} \sum_{y=\lfloor \frac{i}{k} \rfloor+1}^{b} FA^{xy} < f(M)$. The first condition ensures that M occurs more frequently than other characters for a given position i. The second condition ensures that the maximum number of factors intersecting $E(Ref)[i]$ is smaller than $f(M)$. Note that a smaller α means that $f(M) \geq \alpha$ occurs more often, thus implying more frequent rewriting.

Algorithm 2 presents the pseudo-code of rewriting references. If a reference is rewritten, we update its corresponding k-mers (Line 5). Moreover, we store the rewritten reference by referentially representing it according to the corresponding original one. The aim is to reduce the storage needed when introducing a new reference. Thus, we do not consider to rewrite a reference more than once, as it introduces θ-order compression ($\theta > 2$) at the cost of decreased efficiency of decompression and querying. This is also the reason that we delete **FA** and rp after rewriting a reference (Line 6). In addition, we observe that the factor matrix **FA** of $E(Ref)$ still takes up unnecessary space due to FA^{xy} ($y > x$) $= \emptyset$. Hence, to store FA^{xy}, we construct a vector FV of size $\frac{b \cdot (b+1)}{2}$, where $b = \lceil \frac{|E(Ref)|_r}{k} \rceil$ and $|E(Ref)|_r$ is the length of $E(Ref)$ that is used as a reference. FV is enlarged over time according to the latest $|E(Ref)|_r$. The worst case time complexity of reference rewriting for a reference is $O(|FV|)$. However, Lemma 2 enables rewriting by scanning part of FV (as shown in Example 7), which enhances the efficiency of reference rewriting. This is also studied in Section 7.

4.6 Reference Selection and Deletion for $V(Tr^n)$

The reference selection for $V(Tr^n)$ is almost the same as that for $E(Tr^n)$. However, as a speed $V(Tr^n)[i]$ is a float, we cannot directly apply k-mer matching to it. Instead, we convert it to an integer $\ddot{V}(Tr^n)[i]$, where $\ddot{V}(Tr^n)[i]$ is the integer closest to $\frac{V(Tr^n)[i]}{0.5^{\eta}}$. This strategy is consistent with the error-bounded referential representation in Section 4.2. The reference deletion for $V(Tr^n)$ is also similar to that for $E(Tr^n)$; thus, we omit it. Moreover, we do not rewrite a reference $V(Ref)$, as speed patterns may vary substantially during different time periods [12, 24] and the latest patterns have already been mined by reference selection.

5 COMPRESSION

We proceed to present the binary encoding and data transmission of streaming trajectories.

5.1 Binary Encoding

We denote the binary code of seq as $\widehat{seq}$. e.g., the binary code of $E(\cdot)$ is denoted as $\hat{E}(\cdot)$.

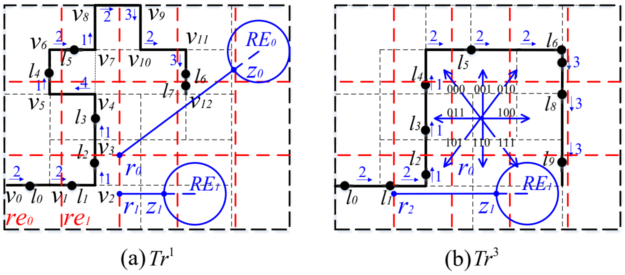
Binary Encoding of References. We follow UTCQ [19] to compress $E(Ref)$. Moreover, we adapt a typical scheme [41] to encode $RD(Ref)$ and $GD(Ref)$. Specifically, given an encoding error bound γ, the binary code $\hat{fv}$ of a floating number fv is calculated as $\hat{fv} = \arg\min_{\hat{fv}_m} \left|\sum_{i=1}^{|\hat{fv}_m|} \hat{fv}_m[i-1] \cdot \frac{1}{2^i} - fv\right|$, where $|\hat{fv}_m| = \gamma$. For example, given $fv = 0.37$ and $\gamma = 3$, we get $\hat{fv}$=011, i.e., fv is approximated as 0.375. The binary code of a reference $\phi(Ref)$ ($\phi = E, GD$) is delivered to the centralized cloud immediately after it is generated.

Binary Encoding of Non-references. We set the lengths of the binary codes of both S and L the same, denoted as len, and record it for decoding a factor (ref_{id}, S, L, M) in a streaming setting. The Exp-Golomb encoding [37] is adopted to compress len. Moreover, len is set to 0 if the reference $\phi(Ref)$ has terminated before generating a factor (ref_{id}, S, L, M). This way, we can improve compression as $len = 0$ only takes 1 bit, and both $\hat{S}$ and $\hat{L}$ are still decodable as the length of $\phi(Ref)$ is known [19]. Finally, $\hat{M}$ takes $\lceil \log_2 o \rceil$ bits for $Com_E(Nref)$ and takes γ bits for $Com_V(Nref)$, where o is the maximum number of outgoing edges for any vertex $v \in V$.

5.2 Transmission of Compressed Binary Codes

As illustrated in Section 4.3, a factor (ref_{id}, S, L, M) cannot be generated until the mis-matched character M is found. It is easy to recognize each part of a factor if we transmit it as a whole to the centralized cloud. However, in this case, the centralized query processor is unable to receive the latest data in real-time, resulting in inaccurate results. To avoid this, we continue transmitting the up-to-date information of a factor during its formation.

We propose a data transmission strategy, which targets low transmission overhead and enables decoding at the centralized cloud without the need of delivering extra information. Given the value of k for k-mer, the transmission of a factor (ref_{id}, S, L, M) is completed in three states: ① transferring the initialized factor $(ref_{id}, S, L', \emptyset)$, where $L' \geq k$ and $|\hat{S}| = |\hat{L'}|$; ② transferring the updated L'; and ③ transferring the mis-matched element M when L' is updated to L. We denote the binary code of a factor transmitted to the centralized cloud at each timestamp as $\hat{bc}$. Obviously, $|\hat{bc}| = |\widehat{ref}_{id}| + 2 \cdot |\hat{S}| \geq |\widehat{ref}_{id}| + 2 \cdot k$ at step ①, and the $\hat{bc}$ that is used to update L' at step ② is always $|\widehat{ref}_{id}| + 1$. As the maximum number of outgoing edges for any vertices in a road network is generally no less than 4, i.e, $o \geq 4$, and we set $\gamma \geq 2$, we have $|\hat{bc}| = |\widehat{ref}_{id}| + |\hat{M}| > |\widehat{ref}_{id}| + 1$ at step ③. If we let $2 \cdot |\hat{S}| \neq |\hat{M}|$, the binary codes transferred during the above three states can be distinguished just by $|\hat{bc}|$. The initial state is ③.



(a) Tr^1 (b) Tr^3

Figure 5: The partition of the road network G in Figure 2.

Example 8. *Continuing Example 5 and letting $\widehat{ref}_{id}$ take 3 bits, the first factor of $Com_E(Tr^6)$ is initialized as $(1, 0, 3, \emptyset)$ and thus is encoded as $(000, 00, 10)$, i.e., $\hat{bc}$ = 0000010. Then it is sent to the centralized cloud, which triggers the state transition from ③ to ①. Next, we continue to transfer $\hat{bc}$ = 0001 before the mis-matched value $M = 2$ is found, during which the state is first transferred to ② and then remains unchanged. Meanwhile, L' continues to be incremented by 1. Once the centralized cloud receives $M = 2$, i.e., $\hat{bc}$ = 000001, a factor is generated and stored in the form $(\hat{M}, \hat{len}, \widehat{ref}_{id}, \hat{S}, \hat{L})$, i.e., $(001, 0, 000, 0000, 1000)$, in the centralized cloud, where we assume $o = 7$, i.e., $\hat{M}$ takes 3 bits.*

6 QUERY PROCESSING

6.1 Query Definition

Definition 10. *Given a query region RE and a set of compressed streaming trajectories $\hat{\mathbf{Tr}}$, a **range query** $\mathbf{range}(\hat{\mathbf{Tr}}, RE)$ returns the set of streaming trajectories $Tr^n (1 \leq n \leq N)$ in $\mathbf{Tr}$, such that $Tr^n \cap RE \neq \emptyset$ at the current timestamp.*

We denote the lastest timestamp of a trajectory Tr^n as $Tr^n.tp$, i.e., $Tr^n.tp = t(Tr^n)[|t(Tr^n)| - 1]$.

Example 9. *Considering the streaming trajectories in Figure 2 and assuming that the current timestamp is 7:04:06, we have $Tr^1.tp$ =7:04:06 and $Tr^3.tp$ =7:03:55. Given $k = 2$, $\gamma = 7$, $\eta = 0$, and $|(v_0 \rightarrow v_1)| = |(v_1 \rightarrow v_2)| = |(v_2 \rightarrow v_3)| = 100$, we get $ad(Tr^1)[2] = 254.76$, $ad(Tr^3)[1] = 127.38$, and $V(Tr^3)[1] = 5.12$ after decompression, i.e., Tr^1 is located on $(v_2 \rightarrow v_3)$ and Tr^3 is located on $(v_1 \rightarrow v_2)$; thus, the range query $\mathbf{range}(\hat{\mathbf{Tr}}, RE_1)$ returns $\emptyset$.*

A naive strategy for computing range queries over compressed streaming trajectories is to decompress each $Tr^n (1 \leq n \leq N)$ and calculate their current locations, which is time-consuming. Instead we introduce indexing and filtering to achieve fast query processing.

6.2 Index and Filtering Technique

We partition the road network G using grid cells re_m, each of which links to the streaming trajectories that are currently located in it. The number of grid cells is denoted by gc. Figure 5 partitions the road network G in Figure 2. Considering the examples in Figure 2 and assuming the current timestamp is 7:03:25, re_0 links to Tr^1.

Definition 11. *The **minimum distance** $mind(re_m, RE)$ between a grid cell re_m and a query region RE is the distance between a location r and a location z, denoted as $|rz|$, where $r \in re_m \wedge z \in RE \wedge \forall r' \neq r\ (r' \in re_m \wedge |r'z| \geq |rz|) \wedge \forall z' \neq z\ (z' \in RE \wedge |rz'| \geq |rz|)$.*

For instance in Figure 5a, the minimum distance between grid cell re_1 and query region RE_0 is $|r_0z_0|$. Since queries are computed centrally, we need to calculate the current location of each streaming trajectory from the most recently arrived data. Given a speed constraint μ of the road network and a timestamp tc, the **reachable distance** of a streaming trajectory Tr^n w.r.t. tc, denoted as dis, is $(tc - Tr^n.tp) \cdot \mu$. Following Example 9 and given $\mu = 21$, the reachable distance of Tr^1 w.r.t. 7:04:06 is $0 \times 21 = 0$, while that of Tr^3 w.r.t. 7:04:06 is $11 \times 21 = 231$.

Lemma 3. *Given a range query* $\boldsymbol{range}(\hat{\mathbf{Tr}}, RE)$*, the current timestamp* tc*, the reachable distance* dis *of* Tr^n *w.r.t.* tc*, and* Tr^n *located in grid cell* re_m *at* $Tr^n.tp$*, if the minimum distance* $mind(re_m, RE) > dis$*,* Tr^n *cannot be in the result.*

Proof. Assuming that Tr^n has a mapped location l at $Tr^n.tp$, we have $l \in re_m$ as Tr^n located in grid cell re_m at $Tr^n.tp$. Since $mind(re_m, RE) > dis$, the distance between l and RE must also exceed the reachable distance dis of Tr^n w.r.t. tc, i.e., Tr^n cannot reach RE at tc. Hence, Tr^n cannot be in the query result. □

Lemma 3 enables pruning Tr^n without computing its location at the current timestamp. Following Example 9 and given $|r_2z_1| = 233$ in Figure 5b, we do not need to decompress $\hat{Com}_E(Tr^3)$ and $\hat{Com}_V(Tr^3)$ if a range query $\boldsymbol{range}(\hat{\mathbf{Tr}}, RE_1)$ arrives at 7:04:06. This is because Tr^3 overlaps re_0 at 7:03:55 and the reachable distance of Tr^3 w.r.t. 7:04:06 is 231<233. If Tr^n cannot be filtered, we need to fully decompress it to calculate its current location. The details are omitted due to the space limitation.

6.3 Index Transmission

Index information is created at the edge server once new data arrives, while query processing occurs centrally. Hence, we need to deliver the index to the centralized cloud. As illustrated in Section 5.2, we always transmit a compressed trajectory once it is (referentially) represented, in order to support accurate queries. A naive strategy is to transfer the ID of the grid cell where Tr^n is located at each timestamp, which incurs high transmission cost.

We propose to transfer the ID of the grid cell, denoted as g_{id}, for Tr^n only when it changes. As shown in Figure 5b, Tr^n can enter at most 8 grid cells if it leaves the current grid cell. Therefore, $\hat{g_{id}}$ takes 3 bits and is appended to $\hat{bc}$. We add one bit at the beginning of $\hat{bc}$ to identify whether it carries index information. The transmission algorithm that includes indexes is very similar to that in Section 5.2, so we omit the details due to the space limitation.

6.4 Discussion

TRACE is able to adopt and support a variety of partitioning methods and queries, as discussed next.

Road Network Partition. We introduce two representative partitioning methods, spatial partitioning [22, 30] and graph-based partitioning [1, 35, 36], which are alternatives to our grid partitioning. Quad-tree partitioning [22, 30] is used often in different settings. It recursively decomposes the space while considering the spatial distribution of the underlying data, and it stops when some pre-defined conditions are satisfied. Each tree node represents a subregion and has either exactly four children (an internal node), or no children (a leaf node). In congestion-based partitioning of a road network [1, 35, 36], edges are associated with feature values and traffic densities. Based on this information, different heterogeneous subregions of a road network are identified that exhibit homogeneous traffic congestion patterns internally.

Table 3: Trajectory datasets.

Datasets	Storage of NCTs	# of NCTs	# of edges per NCT
Denmark	3.47 GB	415,920	Average 95.844
Hangzhou	24.60 GB	1,918,677	Average 250.540
Synthetic	384.61 GB	50,000,000	Average 137.712

Table 4: Parameter ranges and default values.

Parameter	Range
the length of k-mer	5, 7, **9**, 10, 11, 13, **15**, 20, 25
the value of C	0.1, 0.3, 0.5, 0.7, **0.9**
the value of α	**2**, 3, 4, 5, **6**, 8, ∞
the number of grid cells gc	8^2, 16^2, 32^2, $\mathbf{64^2}$, 128^2

The above-mentioned partitioning methods and corresponding indexing techniques can be adapted straightforwardly to TRACE, by considering subregions as grid cells. We use grid partitioning because it is simple, is very efficient, and has low construction cost [24]. However, exploration of possible benefits of other partitioning techniques is a relevant topic for future work.

Query. We introduce two different types of queries that can be supported by TRACE, i.e., a shortest path query and a KNN query. We use l_c to denote the mapped GPS point of a streaming trajectory Tr^n at the current timestamp t_c.

Shortest path query: $\boldsymbol{short}(\hat{Tr^n}, v_q)$. Given a vertex v_q and a compressed streaming trajectory $\hat{Tr^n}$, $\boldsymbol{short}(\hat{Tr^n}, v_q)$ returns the shortest path distance between l_c and v_q. To answer $\boldsymbol{short}(\hat{Tr^n}, v_q)$, we first decompress $\hat{T}(Tr^n)$, $\hat{E}(Tr^n)$ and $\hat{GD}(Tr^n)$ (or $\hat{V}(Tr^n)$) to get $l_c = ((v_i \rightarrow v_j), nd(v_i, l_c), t_c))$. Then the shortest path distance between l_c and v_q is obtained by computing the shortest path distances between v_i and v_q and between v_j and v_q [15]. This process can be facilitated by constructing a G*-tree [23].

KNN query: $\boldsymbol{KNN}(\hat{Tr^n}, \hat{\mathbf{Tr}}, \Bbbk)$. Given a threshold $\Bbbk$, a compressed streaming trajectory $\hat{Tr^n}$, and a set of compressed streaming trajectories $\hat{\mathbf{Tr}}$, $\boldsymbol{KNN}(\hat{Tr^n}, \hat{\mathbf{Tr}}, \Bbbk)$ returns the top $\Bbbk$ streaming trajectories in $\hat{\mathbf{Tr}}$ ranked by their shortest path distances to l_c at t_c in ascending order. The process of computing $\boldsymbol{KNN}(\hat{Tr^n}, \hat{\mathbf{Tr}}, \Bbbk)$ is similar to that of computing $\boldsymbol{short}(\hat{Tr^n}, v_q)$. In addition, a priority queue is maintained to perform the most promising vertex expansions [23].

7 EXPERIMENTAL EVALUATION

7.1 Experimental Setting

Datasets. We use two real datasets, Denmark (DK) and Hangzhou (HZ), and a synthetic dataset, Synthetic (Syn), as described in Table 3. DK is collected from 162 vehicles over about 2 years from Jan. 2007 to Dec. 2008 in Denmark, while HZ is collected from 24,515 taxis during Nov. 2011 in Hangzhou, China. Syn is generated using the road network of Hangzhou. It contains five data groups, each with the same number of trajectories (i.e., Syn0.1, Syn0.3, Syn0.5, Syn0.7, and Syn0.9), where the similarities between each pair of trajectories are 0.1 ± 0.05, 0.3 ± 0.05, 0.5 ± 0.05, 0.7 ± 0.05, and 0.9 ± 0.05, respectively. The similarity is measured according to the Longest Common Road Segment (LCRS) [42]. We generate similar speed patterns when trajectories traverse an LCRS. Specifically, any two subsequences of speeds corresponding to an LCRS, denoted as $sub(V^n)$ and $sub(V^{n'})$, satisfy $|sub(V^n)| = |sub(V^{n'})| \wedge |sub(V_i^n) - sub(V_i^{n'})| \leq \frac{1}{2}^{\eta}$ $(0 \leq i < |sub(V^n)|)$, where $\eta = 7$. The default sample interval of HZ and Syn is 20s, and that of DK is 1s.

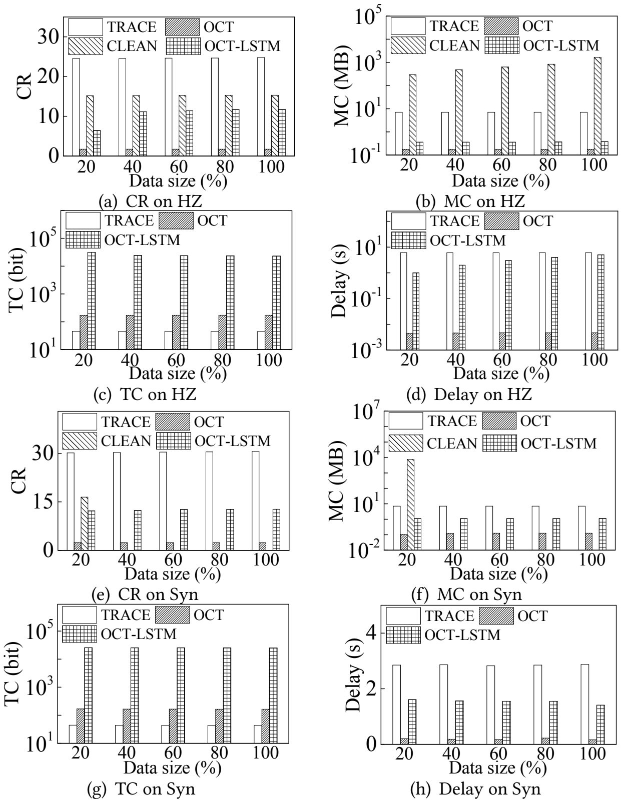
Parameter Setting. In the experiments, we study the effect on the performance of the parameters summarized in Table 4. We set λ to 0.998 and γ to 3 on both datasets, and set η to 3 on DK and 7 on HZ, respectively. Note that the default values of parameters for Syn are the same as those for HZ. The cardinalities of the hash tables for storing $E(\mathbf{Ref})$ and $V(\mathbf{Ref})$, are both 1000. All algorithms are implemented in C++ and run on a computer with an Intel Core i9-9880H CPU (2.30 GHz) and 32 GB memory.

Comparison Algorithms. We compare TRACE with three methods: CLEAN [44], OCT-LSTM [5], and OCT [5]. CLEAN is an offline method, while OCT-LSTM trains an LSTM model to obtain repetitive patterns of time-distance sequences using historical data. OCT-LSTM compresses time-distance sequences by discarding data if the prediction deviation is smaller than an error bound. OCT [5] is similar to OCT-LSTM, except that it uses a linear model for prediction and does not perform any offline training.

Performance Metrics. For compression, we use the compression ratio (CR), time delay (Delay), maximum memory cost (MC), and transmission cost (TC) as the performance metrics. For query processing, we use the query time (Time) and transmission cost (TC) as the performance metrics. Specifically, the maximum memory cost records the maximum storage of auxiliary structures (such as hash tables) created for compression over all timestamps. The auxiliary structures are stored in main memory at the edge server, while the compressed data is transmitted to and stored in the centralized cloud. Moreover, both the time delay and transmission cost are reported as average values to process an arriving mapped GPS point at a timestamp. As the experiments are simulated in a vehicular edge computing architecture, the transmission cost is the size of contents to be transferred from the edge server to the centralized cloud. The contents include compressed trajectories and indexes, thus we use the transmission cost as a performance metric of both compression and query processing.

7.2 Experimental Results

Comparison and Scalability. Figure 6 reports experimental findings when varying the dataset size from 20% to 100%. We use 80%, 10%, and 10% of each dataset for training, validation, and testing for OCT-LSTM, respectively. Thus, all methods are applied to 10% of each dataset in this set of experiments. Since CLEAN is an offline method and takes 192.1 hours to compress 20% Syn, we only report its compression ratio and maximum memory cost on HZ and 20% Syn. First, TRACE outperforms all the baselines in terms of compression ratio and transmission cost. This is mainly due to TRACE's separate compression of paths, speeds, and timestamps that eliminates more redundancy. Second, the compression ratios of TRACE and OCT-LSTM increase slightly, as more subsequences can be referentially compressed and more training data is used. The transmission cost of OCT-LSTM is two orders of magnitude higher than that of TRACE. The reason is that OCT-LSTM needs to update and transmit the model to adapt to new speed patterns. Next, the maximum memory cost and time delay of TRACE exceed those of OCT-LSTM and OCT. This is in line with TRACE's goal of achieving high compression ratios without losing mapped GPS points. Hence, TRACE needs to maintain references in main memory and needs to wait for k mapped GPS points to initialize a factor. In contrast, OCT-LSTM and OCT trade the accuracy for compression efficiency and memory cost, by discarding data that can be predicted within an error bound. However, the maximum memory cost of TRACE never exceeds 8 MB, which we expect is easily accommodated by the edge server layer [29]. The maximum memory cost and the processing time of CLEAN are the highest among the four methods. However, CLEAN is the best among the baselines in terms of compression ratio; thus, it is a good option if the compression can take place offline and the dataset is relatively small. Finally, the time delay of TRACE is roughly independent of the dataset size, while that of OCT-LSTM drops slightly with more data as fewer models are re-trained. The time delay of TRACE on both datasets never exceeds its corresponding default sample interval, i.e., 20s, which suggests that TRACE supports real-time compression.



(a) CR on HZ

(b) MC on HZ

(c) TC on HZ

(d) Delay on HZ

(e) CR on Syn

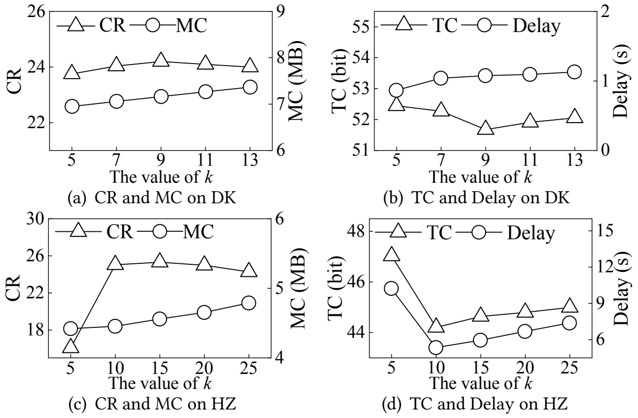
(f) MC on Syn

(g) TC on Syn

(h) Delay on Syn

Figure 6: Comparison and scalability.

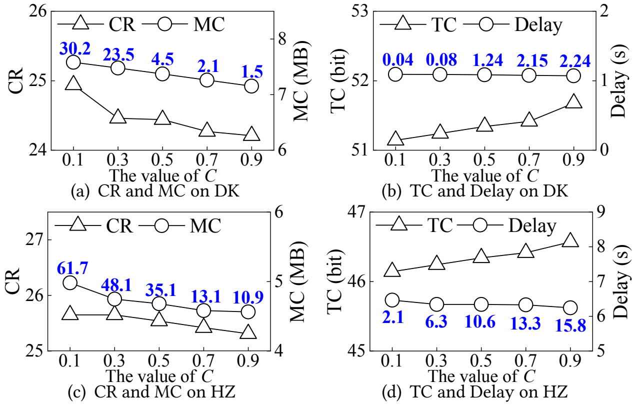
Effect of k. Figure 7 reports the results when varying k, the length of k-mers. First, the compression ratios on both DK and HZ first increase and then drop. On one hand, a larger k results in more references due to the higher probability of mismatching among longer sequences, which reduces the compression ratio. On the other hand, a longer subsequence tends to be encoded into one factor, increasing the compression ratio. Second, the maximum memory cost increases with k, because more references lead to more k-mers being stored. Next, the transmission cost has an opposite trend to the compression ratio. The higher the compression ratio we can achieve, the less data is delivered to the centralized cloud.



(a) CR and MC on DK (b) TC and Delay on DK

(c) CR and MC on HZ (d) TC and Delay on HZ

Figure 7: Effect of k.



(a) CR and MC on DK (b) TC and Delay on DK

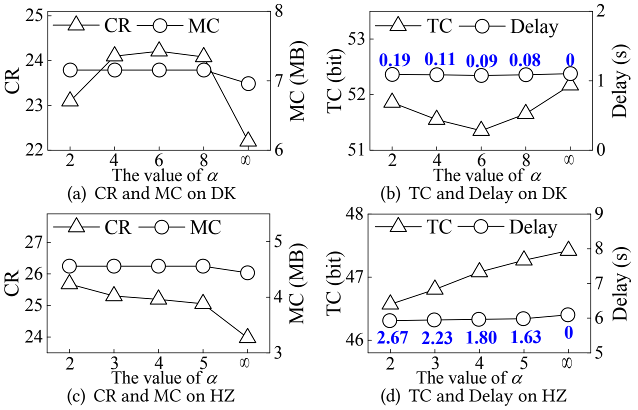
(c) CR and MC on HZ (d) TC and Delay on HZ

Figure 8: Effect of C.

Finally, Figures 7b and 7d show that the time delay increases with k in most cases, as the time to initialize each factor rises.

Effect of C. Figure 8 studies the effect of the deletion coefficient C that controls the amount of outdated data. Specifically, the blue numbers along with the maximum memory cost in Figures 8a and 8c report $\frac{km_{max}}{km}\%$, where km_{max} is the maximum number of the k-mers in memory over all timestamps using reference deletion, while km is the number without using reference deletion. The figures show that reference deletion function reduces the memory cost substantially. We also report the average processing time (ms) of the reference deletion at each timestamp in Figures 8b and 8d (the blue numbers along with the time delay). As observed, the processing time is 2–4 orders of magnitude less than the time delay. In addition, both the compression ratio and the time delay drop with the increasing of C on both DK and HZ. This is because a larger C results in more deletions, which reduces the number of k-mers stored in memory. In this case, an upcoming sequence is more likely to be assigned as a reference. Since the time delay is mainly caused by compressing non-references, it drops with the decrease of non-references. However, the drops are smooth because we only delete k-mers that are unlikely to be referenced.

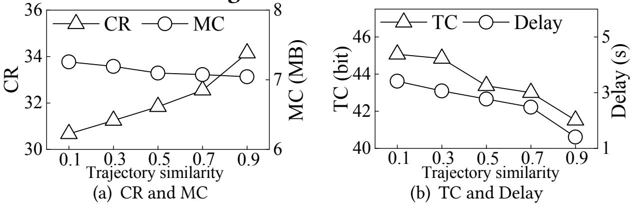
Effect of α. Figure 9 reports the impact of the rewriting coefficient α on the compression performance, where α controls the frequency of rewriting and "∞" indicates no rewriting. We see that the compression ratio of TRACE first increases and then drops with the increase of α on DK, while it continues to increase with α on HZ. The reason is that subsequences may be identified wrongly as being frequent if a small α is used, while many truly frequent subsequences may be missed if a very large α is used. Intuitively, the optimal α value highly depends on the dataset. Second, the maximum memory cost of TRACE drops when $\alpha = \infty$, because we do not maintain any auxiliary structures for rewriting in this case. Next, the time delay of TRACE follows the opposite trend of the compression ratio on both datasets, as it is mainly caused by initializing factors. Finally, the blue numbers along with the time delay in Figures 9b and 9d are the average processing time (ms) of reference rewriting at each timestamp, which are negligible compared with the time delay.



(a) CR and MC on DK (b) TC and Delay on DK

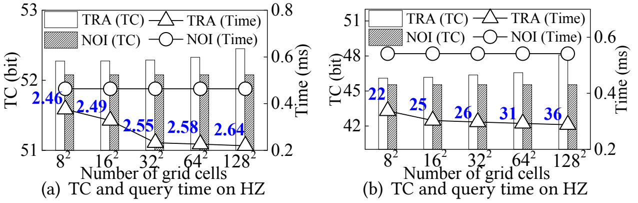
(c) CR and MC on HZ (d) TC and Delay on HZ

Figure 9: Effect of α.



(a) CR and MC (b) TC and Delay

Figure 10: Effect of trajectory similarity.



(a) TC and query time on HZ (b) TC and query time on HZ

Figure 11: Effect of number of grid cells gc.

Effect of Trajectory Similarity. We perform TRACE on Syn0.1, Syn0.3, Syn0.5, Syn0.7, and Syn0.9, denoted as 0.1, 0.3, 0.5, 0.7, and 0.9, respectively, to study the impact of trajectory similarities on compression. Figure 10 shows that the compression ratio increases and that the transmission cost drops as trajectories become more similar. Moreover, the maximum memory cost and the time delay decrease with the increase of similarity because fewer k-mers are stored and fewer factors are generated. Overall, TRACE achieves higher compression performance on datasets with larger similarity, due to its referential compression.

Effect of gc. Figure 11 reports the transmission cost and query time when varying the number of grid cells, gc. "TRA" denotes our TRACE framework while "NOI" denotes the case of no indexing. It is clear that a larger gc results in higher query efficiency and larger transmission cost. The blue numbers along the TRACE query time line denote the index creation time (μs), which is the average time used on creating/updating indexes for all the arriving locations per timestamp. The index creation time increases slightly with a finer grid granularity, i.e., larger gc. This is because TRACE updates the grid information of streaming trajectories once this changes and then delivers the new information to the centralized cloud to improve the query efficiency. However, the grid information contributes at most 6.5% to the total transmission cost and the index creation time is negligible compared with the time delay.

8 RELATED WORK

8.1 Raw Data Compression

Raw trajectory compression aims to compact trajectories that have not been map-matched and targets either offline [6, 26, 45] or online settings [8, 18, 27]. REST [45] is the first offline reference-based raw trajectory compression framework. Targeting raw trajectories, it differs very substantially from TRACE. REST compresses the timestamps of a non-reference w.r.t. to that of a reference only when their spatial information are matchable, while TRACE referentially compresses different parts of trajectories separately, which enables the removal of more redundancy. SQUISH [27] is a representative work that aims at reducing data loss and preserves speed information at high accuracy during compression. Deng et al. [8] consider direction-preserving compression in streaming settings and propose an advanced online DPTS algorithm that achieves high compression ratios. Li et al. [18] take into account the special needs of real-time surveillance applications and increase the loading speed of trajectory data very noticeably. Comprehensive experimental evaluations of raw trajectory compression are available [14, 28, 43].

8.2 Network-constrained Compression

Network-constrained trajectory compression leverages the underlying road network to improve compression, which also occurs either offline or online.

Offline Mode. Krogh et al. [17] compress a trajectory by storing only the first and last edges of each shortest path in a trajectory. Ji et al. [13] encode outgoing road segments clockwise based on a pre-computed clockwise code table. Sun et al. [11, 32] propose a two-stage spatial compression algorithm using shortest path and frequent subtrajectory compression. Yang et al. [41] present a very compact representation that separates the distance information from timestamps. Koide et al. [16] develop a compression technique for spatial information of trajectories and support the retrieval of subpaths. Sui et al. [33] assign each GPS point to the middle point of a segment and propose a road-network partitioning strategy on which the compression ratio depends. CLEAN [44] encodes trajectories by means of frequent patterns. In particular, CLEAN is the first study to perform temporal compression on top of spatial compression and presents novel pattern concatenation and generation techniques that always expand the pattern with the highest support. However, CLEAN needs to count the support of each newly generated pattern in the trajectory dataset. This incurs high main memory and running time costs for large datasets, which precludes it from running in an online manner. UTCQ [19] targets compression of uncertain trajectories; in contrast, TRACE aims to compress streaming trajectories in real-time. Specifically, UTCQ uses an improved TED representation and develops an FJD function to measure the similarity between trajectory instances and to select references in batch mode. Instead, TRACE proposes a more compact speed-based representation and adapts k-mer matching for real-time reference selection.

Online Mode. Four studies exist that target online network-constrained trajectory compression. Chen et al. [3, 4] present a solution that compresses the edge sequences in trajectories by retaining only out-edges with remarkable heading changes and also uses frequent paths trained offline to further improve compression. The solution does not consider the compression of temporal information and locations of trajectories. ONTRAC [31] uses a k-order Markov model to learn frequent paths and apply them to compress incoming edges in real-time. This solution discards the mapped GPS points of the original trajectories and only estimates the time of traversing an edge using a trained model. OCT-LSTM [5] trains models to obtain repetitive movement patterns of the time-distance sequences using historical data and only transfers data when predicted values deviate significantly from the actual ones. Then, re-training is performed at the centralized cloud and the updated model is transmitted to the edge server layer, incurring a high transmission cost. Moreover, OCT-LSTM does not mine frequent paths that exist widely and can enhance compression substantially. If no historical data exists, OCT-LSTM uses linear prediction, called OCT. All the above methods compress trajectories by discarding useful information, but TRACE keeps all of them to maintain data usability. Moreover, both OCT-LSTM and ONTRAC employ an offline training phase as the foundation for their online compression, while TRACE is designed to compress streaming trajectories in a fully online fashion, making it more generally usable in practice.

9 CONCLUSION AND FUTURE WORK

We propose a new framework for compressing, transmitting, and querying streaming network-constrained trajectories in real-time. We develop a speed-based and a multiple-references based referential representation to represent trajectories concisely. We propose k-mer matching based online reference selection with reference deletion and rewriting. Deletion reduces the storage cost, while rewriting improves the compression ratio. Moreover, we propose a transmission strategy that reduces the transmission cost and ensures that compressed trajectories are decodable. Finally, we enable querying of compressed streaming trajectories and provide indexing and filtering techniques that accelerate real-time query processing. An experimental study using two real-life datasets and one synthetic dataset shows that the proposed TRACE framework outperforms three baselines [5, 44] in terms of compression ratio and transmission cost. In future research, it is of interest to reduce the latency and memory cost of online compression and to deploy TRACE in a distributed cloud setting.

ACKNOWLEDGMENTS

This work was supported in part by the DiCyPS and DIREC centers, funded by Innovation Fund Denmark. Lu Chen is the corresponding author.

REFERENCES

[1] Tarique Anwar, Chengfei Liu, Hai L Vu, Md Saiful Islam, and Timos Sellis. 2018. Capturing the spatiotemporal evolution in road traffic networks. *TKDE* 30, 8 (2018), 1426–1439.

[2] Michel Bierlaire, Jingmin Chen, and Jeffrey Newman. 2013. A probabilistic map matching method for smartphone GPS data. *TRANSPORT RES C-EMER* 26 (2013), 78–98.

[3] Chao Chen, Yan Ding, Zhu Wang, Junfeng Zhao, Bin Guo, and Daqing Zhang. 2019. VTracer: When online vehicle trajectory compression meets mobile edge computing. *IEEE Syst J* 14, 2 (2019), 1635–1646.

[4] Chao Chen, Yan Ding, Xuefeng Xie, Shu Zhang, Zhu Wang, and Liang Feng. 2019. TrajCompressor: An online map-matching-based trajectory compression framework leveraging vehicle heading direction and change. *IEEE Trans. Intell. Transport. Syst* 21, 5 (2019), 2012–2028.

[5] Jie Chen, Zhu Xiao, Dong Wang, Daiwu Chen, Vincent Havyarimana, Jing Bai, and Hongyang Chen. 2018. Toward Opportunistic Compression and Transmission for Private Car Trajectory Data Collection. *IEEE Sens. J.* 19, 5 (2018), 1925–1935.

[6] Minjie Chen, Mantao Xu, and Pasi Franti. 2012. A fast $o(n)$ multiresolution polygonal approximation algorithm for GPS trajectory simplification. *TIP* 21, 5 (2012), 2770–2785.

[7] Yixin Chen and Li Tu. 2007. Density-based clustering for real-time stream data. In *SIGKDD*. 133–142.

[8] Ze Deng, Wei Han, Lizhe Wang, Rajiv Ranjan, Albert Y Zomaya, and Wei Jie. 2017. An efficient online direction-preserving compression approach for trajectory streaming data. *Future Gener Comput Syst* 68 (2017), 150–162.

[9] Sebastian Deorowicz and Szymon Grabowski. 2011. Robust relative compression of genomes with random access. *Bioinformatics* 27, 21 (2011), 2979–2986.

[10] Anton Dignös, Michael H Böhlen, and Johann Gamper. 2014. Overlap interval partition join. In *SIGMOD*. 1459–1470.

[11] Yunheng Han, Weiwei Sun, and Baihua Zheng. 2017. COMPRESS: A comprehensive framework of trajectory compression in road networks. *TODS* 42, 2 (2017), 11.

[12] Gang Hu, Jie Shao, Fenglin Liu, Yuan Wang, and HengTao Shen. 2016. If-matching: Towards accurate map-matching with information fusion. *TKDE* 29, 1 (2016), 114–127.

[13] Yudian Ji, Yuda Zang, Wuman Luo, Xibo Zhou, Ye Ding, and Lionel M Ni. 2016. Clockwise compression for trajectory data under road network constraints. In *ICBDA*. 472–481.

[14] Jonathan Muckell, Jeong-Hyon Hwang, Catherine T. Lawson and SS Rav. 2010. Algorithms for compressing GPS trajectory data: an empirical evaluation. In *SIGSPATIAL*. 402–405.

[15] Ken CK Lee, Wang-Chien Lee, Baihua Zheng and Yuan Tian. 2012. ROAD: A new spatial object search framework for road networks. *TKDE* 24, 3 (2012), 547.

[16] Satoshi Koide, Yukihiro Tadokoro, Chuan Xiao, and Yoshiharu Ishikawa. 2018. CiNCT: Compression and retrieval for massive vehicular trajectories via relative movement labeling. In *ICDE*. 1097–1108.

[17] Benjamin Krogh, Christian S Jensen, and Kristian Torp. 2016. Efficient in-memory indexing of network-constrained trajectories. In *SIGSPATIAL*. 17–26.

[18] Lin Li, Xuezhi Xia, and Ziqian Xiong. 2019. A Novel Online Trajectory Compression Algorithm for Real-time Trajectory Surveillance Applications. In *IMCEC*. 995–999.

[19] Tianyi Li, Ruikai Huang, Lu Chen, Christian S Jensen, and Torben Bach Pedersen. 2020. Compression of uncertain trajectories in road networks. *PVLDB* 13, 7 (2020), 1050–1063.

[20] Yushuai Li, David Wenzhong Gao, Wei Gao, Huaguang Zhang, and Jianguo Zhou. 2020. Double-mode energy management for multi-energy system via distributed dynamic event-triggered Newton-Raphson algorithm. *IEEE T Smart Grid.* 11, 6 (2020), 5339–5356.

[21] Yushuai Li, Wenzhong Gao, Wei Gao, Huaguang Zhang, and Jianguo Zhou. 2020. A distributed double-Newton descent algorithm for cooperative energy management of multiple energy bodies in energy internet. *IEEE T IND INFORM.* (2020).

[22] Yanhong Li, Guohui Li, Jianjun Li, and Kai Yao. 2018. SKQAI: A novel air index for spatial keyword query processing in road networks. *Inf. Sci.* 430 (2018), 17–38.

[23] Zijian Li, Lei Chen, and Yue Wang. 2019. G*-tree: An efficient spatial index on road networks. In *ICDE*. 268–279.

[24] Wei Liu, Yu Zheng, Sanjay Chawla, Jing Yuan, and Xie Xing. 2011. Discovering spatio-temporal causal interactions in traffic data streams. In *SIGKDD*. 1010–1018.

[25] Yuansheng Liu, Hui Peng, Limsoon Wong, and Jinyan Li. 2017. High-speed and high-ratio referential genome compression. *Bioinformatics* 33, 21 (2017), 3364–3372.

[26] Cheng Long, Raymond Chi-Wing Wong, and HV Jagadish. 2014. Trajectory simplification: on minimizing the direction-based error. *PVLDB* 8, 1 (2014), 49–60.

[27] 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*. 1–8.

[28] Jonathan Muckell, Paul W Olsen, Jeong-Hyon Hwang, SS Ravi, and Catherine T Lawson. 2013. A framework for efficient and convenient evaluation of trajectory compression algorithms. In *COM.Geo*. 24–31.

[29] Salman Raza, Shangguang Wang, Manzoor Ahmed, and Muhammad Rizwan Anwar. 2019. A survey on vehicular edge computing: architecture, applications, technical issues, and future directions. *IEEE Wirel Commun* 19, 4 (2019), 2322–2358.

[30] Sankaranarayanan Jagan Samet Hanan and Alborzi Houman. 2008. Scalable network distance browsing in spatial databases. In *SIGMOD*. 43–54.

[31] Arlei Silva, Ramya Raghavendra, Mudhakar Srivatsa, and Ambuj K Singh. 2016. Prediction-based online trajectory compression. *arXiv preprint arXiv:1601.06316* (2016).

[32] Renchu Song, Weiwei Sun, Baihua Zheng, and Yu Zheng. 2014. PRESS: A novel framework of trajectory compression in road networks. *PVLDB* 7, 9 (2014), 661–672.

[33] Peipei Sui and Xiaoyu Yang. 2018. A privacy-preserving compression storage method for large trajectory data in road network. *J. Grid Comput.* 16, 2 (2018), 229–245.

[34] Shun Taguchi, Satoshi Koide, and Takayoshi Yoshimura. 2018. Online map matching with route prediction. *TITS* 20, 1 (2018), 338–347.

[35] Tarique Anwar, Chengfei Liu, Hai L. Vu and Christopher Leckie. 2014. Spatial Partitioning of Large Urban Road Networks. In *EDBT*. 343–354.

[36] Tarique Anwar, Chengfei Liu, Hai L. Vu and Md Saiful Islam. 2016. Tracking the evolution of congestion in dynamic urban road networks. In *CIKM*. 2323–2328.

[37] Jukka Teuhola. 1978. A compression method for clustered bit-vectors. *INFORM PROCESS LETT* 7, 6 (1978), 308–311.

[38] Sebastian Wandelt and Ulf Leser. 2012. Adaptive efficient compression of genomes. *ALGORITHM MOL BIOL* 7, 1 (2012), 30.

[39] Sebastian Wandelt and Ulf Leser. 2013. FRESCO: Referential compression of highly similar sequences. *TCBB* 10, 5 (2013), 1275–1288.

[40] Yang Xiang, Wanlei Zhou, and Minyi Guo. 2008. Flexible deterministic packet marking: An IP traceback system to find the real source of attacks. *IEEE Trans Parallel Distrib Syst* 20, 4 (2008), 567–580.

[41] Xiaochun Yang, Bin Wang, Kai Yang, Chengfei Liu, and Baihua Zheng. 2017. A novel representation and compression for queries on trajectories in road networks. *TKDE* 30, 4 (2017), 613–629.

[42] Haitao Yuan and Guoliang Li. 2019. Distributed in-memory trajectory similarity search and join on road network. In *ICDE*. 1262–1273.

[43] Dongxiang Zhang, Mengting Ding, Dingyu Yang, Yi Liu, Ju Fan, and Heng Tao Shen. 2018. Trajectory simplification: an experimental study and quality analysis. *PVLDB* 11, 9 (2018), 934–946.

[44] Peng Zhao, Qinpei Zhao, Chenxi Zhang, Gong Su, Qi Zhang, and Weixiong Rao. 2019. CLEAN: frequent pattern-based trajectory spatial-temporal compression on road networks. In *MDM*. 605–610.

[45] 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 *SIGKDD*. 2797–2806.