DAVT: An Error-Bounded Vehicle Trajectory Data Representation and Compression Framework

Chao Chen, Yan Ding, Suiming Guo, and Yasha Wang

***Abstract*—An increasing number of vehicles are now equipped with GPS devices to facilitate fleet management and send their GPS locations continuously, generating a huge volume of trajectory data. Sending and storing such vehicle trajectory data cause sustainable communication and storage overheads. Trajectory data compression becomes a promising way to alleviate overhead issues. However, previous solutions are commonly carried out at the side of the data center after data having been received, thus saving the storage cost only. Here, we bring the idea of mobile edge computing and transfer the computation-intensive data compression task to the mobile devices of drivers. As a result, the trajectory data is reduced at the side of data generators before being sent out; thus, it can lower data communication and storage costs simultaneously. We propose `DAVT`, an error-bounded trajectory data representation, and a compression framework. Specifically, the trajectory data is reformatted into three parts (i.e., `D`istance, `A`cceleration & `V`elocity, and `T`ime), and three compressors are wisely devised to compress each part. For `D` and `AV` parts, a similar *Huffman tree-forest* structure is exploited to encode data elements effectively, but with quite different rationales. For the `T` part, the large absolute timestamps are transformed to small time intervals firstly, and different encoding techniques are adopted based on the data quality. We evaluate our proposed system using a large-scale taxi trajectory dataset collected from the city of Beijing, China. Our results show that our compressors outperform other baselines.**

***Index Terms*—Huffman forest, huffman tree, mobile edge computing, time-varying velocity, trajectory data compression.**

I. Introduction

RECENT years have witnessed an increasing number of GPS-enabled vehicles, including taxis, buses, logistical vans, and so on, thanks to the popularity and maturity of GPS devices and the mobile Internet [3], [7], [16], [33]. They are generating a huge volume of GPS trajectory data, causing sustainable communication and storage costs. One of the main purposes of installing GPS devices in vehicles is fleet management, i.e., knowing their real-time positions for monitoring, tracking, scheduling and so on [8], [23]. Consequently, information needs to be transmitted to the data center in a real-time manner, including the spatial coordinates and velocity of each vehicle. Vehicles usually report their locations to the data center at a higher frequency than necessary to ensure management quality, not only incurring significant communication and storage overheads but also aggravating the computation burdens at the side of the data center. More specifically, map-matching is always applied to identify the "true" positions of vehicles, because the raw data can be far more accurate due to positioning errors of GPS devices [13]. More computation resources are required to guarantee a timely response since more data are sent to the data center [4], [6].

Common data reduction solutions include simply reporting locations less frequently or discarding some locations before sending them to the data center. Such naive methods are problematic in real applications since, 1) the *distance gap* between two retained consecutive points may be too big to infer the locations of vehicles in-between [5], [35]; 2) the *positioning quality* of the retained points can be quite random, making fleet management (e.g., tracking) extremely challenging. To avoid these issues, based on the collected data, current data reduction is often implemented by data compression at the side of data centers (i.e., *centralized*) to lower the storage cost only. To make matters worse, almost all current data reduction approaches neglect the time-varying velocity information directly – such information could be vitally important in understanding the individual driving style, probing road conditions, and so on [3], [8], [14]. Although the average velocity between two consecutive GPS points can be easily estimated, it cannot be viewed as redundant information because it is detected based on Doppler effect [9]. Therefore, the time-varying velocity information should be handled separately.

In this paper, we aim at developing a cost-effective online data compression framework with the idea of *collecting* ***more raw data*** *at the device side but sending* ***less yet still complete data***[1] *to the data center*. Specifically, compared to previous solutions,

Manuscript received March 8, 2020; revised June 27, 2020; accepted August 5, 2020. Date of publication August 10, 2020; date of current version October 22, 2020. This work was supported in part by the National Natural Science Foundation of China under Grant 61872050 and in part by the Chongqing Basic and Frontier Research Program under Grant cstc2018jcyjAX0551. The review of this article was coordinated by Prof. Jun Won Choi. *Chao Chen and Yan Ding contributed equally to this work. (Corresponding author: Chao Chen.)*

Chao Chen is with the Key Laboratory of Dependable Service Computing in Cyber Physical Society (Chongqing University), Ministry of Education and also with the College of Computer Science, Chongqing University, Chongqing 400044, China (e-mail: ivanchao.chen@gmail.com).

Yan Ding is with the State University of New York at Binghamton, NY 13902 USA (e-mail: duke_ding@sina.cn).

Suiming Guo is with the College of Information Science and Technology, Jinan University, Guangzhou 510632, China (e-mail: guosuiming@email.jnu.edu.cn).

Yasha Wang is with the School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China (e-mail: wangyasha@pku.edu.cn).

This article has supplementary downloadable material available at https://ieeexplore.ieee.org, provided by the authors.

Digital Object Identifier 10.1109/TVT.2020.3015214

[1]Here, complete data refers that our compressed data contains the time-varying velocity information, compared with compressed results generated by other data reduction approaches.

our framework presents the following innovative and unique features.

From centralized computing to edge computing. We perform the data compression at the vehicle side (e.g., smartphones) by introducing the idea of mobile edge computing [20], [29] to lower both communication and storage costs. We argue that such a solution is feasible. More specifically, firstly, smartphones of drivers are not only with powerful computation capability but also almost idle during driving. Secondly, data transmission between the GPS devices and smartphones of drivers is highly efficient and free, either by WiFi or Bluetooth. Thirdly, the data compression work previously done at the side of data centers can be easily transferred to smartphones, which is demonstrated in our previous work [4]. As a result, only the reduced data are needed to be sent to the data center instead, incurring less communication and storage overheads.

A new data representation considering velocity. We propose a new data representation that fully considers all the time-varying information. Similar to [1], [24], [30], [37], we also separate the raw trajectory into two components – spatial and temporal trajectories – and perform data compression respectively. In general, the spatial trajectory is represented by a sequence of connected edges in the road network, which records the travelling paths, i.e., the spatial information only; the temporal trajectory is represented by a time-distance curve, which tells the time information along the travelling paths. Spatial trajectory compression is mostly based on the map-matching and is lossless [4], [6], while the temporal trajectory compression is lossy and usually based on the line-simplification [30], [36]. Here, we mainly focus on the temporal trajectory compression since relatively less attention has been paid. More importantly, to the best of our knowledge, the time-varying velocity information is also cared and well represented for the first time. In more detail, we start by seeking a more *accurate and complete* data format and representation and then propose a set of newly error-bounded data compression techniques.

The support of various types of queries. Basically, the real-time locations of vehicles shall be inferred efficiently from the compressed data at the side of data centers, to facilitate tasks including monitoring, tracking, dispatching. Thus, queries including *where* (i.e., the vehicle's position at a given time) and *when* (i.e., at what time that a vehicle located at a given visited location) shall be enabled. Besides, because our new data representation also takes care of the time-varying velocity information, queries like the velocity variance (i.e., acceleration) and traffic conditions of the interested roads at a given time duration can also be supported. Furthermore, more potential spatio-temporal data mining applications can be enabled, e.g., drivers' driving styles or hobbies are embedded implicitly in the time-varying velocity information [3], [8].

From the technical perspective, we mainly make the following contributions in this paper:

- We propose a novel data representation called `DAVT` to reformat a temporal trajectory, which includes three kinds of sequences (i.e., `D`istance, `A`cceleration and `V`elocity, and `T`ime respectively). For each sort of the temporal trajectory, essentially unlike the spatial trajectory compression that achieves data reduction by removing some unnecessary edges or points, we retain all elements of sampling points and artificially produce the redundant information (i.e., repeated values) by value discretization. Afterwards, data compression is fulfilled by the idea of encoding.
- For `D` and `AV` parts, we make use of the *Huffman encoding* to compress each sequence. However, the *Huffman tree* built based on a sequence may not be suitable for encoding another sequence since they have quite different value distributions. To overcome the issue, we propose the idea of offline training *Huffman forest* which consists of a number of *Huffman trees* according to underlying rationales, and complete online element encoding based on the trained Huffman forest for each new incoming sequence. For the `T` part, to achieve a better compression performance, we apply a hybrid of encoding techniques to compress time sequences with different data qualities.
- We extensively conduct a series of experiments to evaluate the performance of the proposed data compression framework. Compared to baselines, the results show that our compressor has not only the best effectiveness in terms of compression ratio, but also good efficiency in both compression and decompression. A compressor with a higher compression ratio can save more communication and storage cost. Moreover, we further theoretically prove the error-boundedness.

The rest of the paper is organized as follows: In Section II, we review the related work and show how this paper differs from previous research. In Section III, we introduce some main concepts and provide an overview of the system. We elaborate details on temporal trajectory data representation, data compression and decompression algorithms in Section IV and Section V, respectively. We evaluate the performance of the proposed framework thoroughly in Section VI. Finally, we conclude the paper and discuss the future research directions in Section VII.

II. Related Work

There are many data compression techniques that have been proposed and widely applied in dealing with different data types, e.g., image, audio, and trajectory data. Interested readers can refer to [32] for a thorough overview. Here, we mainly review the trajectory data compression techniques, which can be broadly categorized into two groups according to their working principles. The first group of techniques mainly works by *discarding* some unnecessary *original* data elements in the trajectory representation, while the second group works by using *new* data elements to *replace/encode* the original one(s) in the trajectory representation.

A. Trajectory Data Compression by Discarding

The working principle of this category is that the discarded data elements can be well or even fully recovered using the remaining data elements in the compressed trajectory. Representative methods include Douglas-Peucker (DP) algorithm as well as its variants [10], PRESS, and so on [30]. For instance, a trajectory in terms of a sequence of GPS points is reduced to the one containing a much fewer number of GPS points. Such GPS points are commonly known as "feature" points. The positions between two consecutive feature GPS points can be approximated by simply linear interpolation. To name a few, authors in [20], [21] propose BQS and its variant FBQS, which achieve online data compression by constructing convex-hulls and discarding GPS points lying inside them. Their computation complexity is $\mathcal{O}(n^2)$, where n is the number of GPS points. OPERB is also an online data compression algorithm but is more efficient, with a computation complexity of $\mathcal{O}(n)$ [19]. Considering vehicles shall move within a limited spatial range (can only move on roads), the trajectory is often represented by a connected node/edge sequence. In the data compression of the spatial trajectory, a node sequence (also known as a path) in which two consecutive nodes are connected by a unique edge in the road network is reduced to a shorter node sequence. In the newly compressed node sequence, any two consecutive nodes are generally disconnected in the road network, but in-between paths can be easily inferred by finding the shortest or the most frequent path. In the data compression of the temporal trajectory, it is usually represented by a sequence of time-distance points in the 2D plane. PRESS determines whether to retain a point according to its contribution to the curve shape, given an acceptable error range. In our recent work and implementation [4], [6], HCC selects a much smaller number of edges in compressing a spatial trajectory. Only out-edges with remarkable driving direction change at intersections are saved. *In summary, each data element in the representation in this category before and after data compression shall have the same physical meaning.*

B. Trajectory Data Compression by Replacement

The working principle of this category is that the newly generated data element occupied much less space than the original one(s). The most well-known representative algorithm is the Huffman coding [15]. The key idea of Huffman coding is to use shorter-length binary codes to encode data elements that appeared more frequently in the trajectory representation. The space occupying by the newly generated element (i.e., binary codes) shall be much smaller, leading to the data reduction. With a similar idea, TED also encodes each data element in the temporal trajectory using the variable-length binary codes [34]. The only difference is that it encodes each data element based on the distance from the head node of the edge. Readers can refer to [34] for more details. In these two cases, the total number of data elements before and after data compression in the trajectory representation are *identical*. Another representative encoding technique is Run-Length Encoding (RLE) [2]. The new data element is a tuple, indicating the data value and its appearance frequency in the trajectory representation, respectively. RLE sometimes fails to work when data redundancy in the representation is little. In this case, the total number of data elements after data compression should be much smaller, contributing to the data reduction eventually. A number of dictionary-based coding approaches are effective in compressing data containing more repeated patterns in the trajectory representation [26], [38]. The new data element is the corresponding index in the built

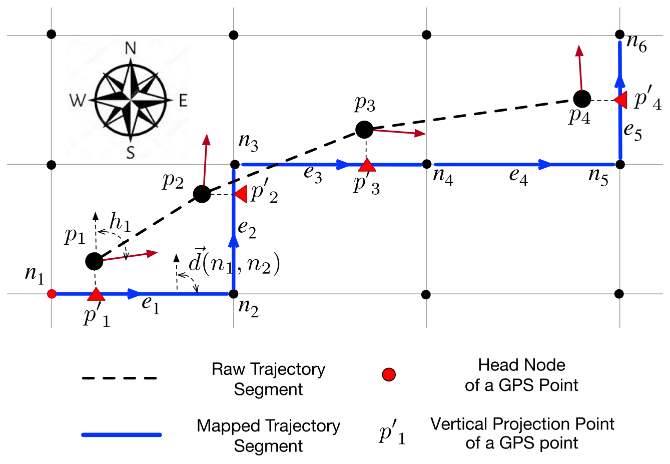


Fig. 1. Illustration of main concepts used in this paper.

dictionary. The data element is also replaced by using some semantic meanings. For example, [28] converts the data element into the street name. [27] replaces it by exploiting the high-level semantic annotation of the transportation network (e.g., bus, tram, and train lines). *In summary, in this category, the physical meaning of each data element in the representation before and after data compression has been changed.* Our work is belonging to this category since both Huffman encoding and RLE are combined and used. But it goes a step further by constructing a set of Huffman trees (i.e., Huffman forest) since raw data element values in trajectories generated in different roads/areas are huge and have quite different value distributions.

III. Main Concepts and System Overview

A. Main Concepts

Definition 1. (Road Network): A road network is a graph $G(N, E)$, consisting of a node set N and an edge set E. Each node n_i in N is represented by a pair of longitude and latitude coordinates, indicating its spatial location. Each edge e_i in E consists of two nodes (i.e., n_a and n_b) and has two edge directions denoted by $\vec{d}(n_a, n_b)$ and $\vec{d}(n_b, n_a)$. $\vec{d}(n_a, n_b)$ stands for the edge direction from node n_a to n_b, which can be easily calculated based on the longitudes and latitudes of both nodes. In this case, n_a is called a head node, while n_b refers to a tail node. For instance, $\vec{d}(n_1, n_2)$ in Fig. 1 represents the edge direction from node n_1 to n_2.

Definition 2. (GPS Entry): A GPS entry records the spatial-temporal information of a moving object. A piece of GPS entry, represented by $r_i = (t_i, x_i, y_i, v_i, h_i)$, generally consists of five elements, i.e., a timestamp t_i, a geospatial coordinate (x_i, y_i), an instantaneous velocity v_i and a heading direction h_i $(0^\circ \leq h_i < 360^\circ)$ using north direction as a basis. For example, h_1 is the vehicle's heading direction at timestamp t_1 shown in Fig. 1. Among these five elements, a pair of timestamp and geospatial coordinate is defined as a GPS point, denoted as $p_i = (t_i, x_i, y_i)$.

Definition 3. (Raw Trajectory Segment): A raw trajectory segment Tr is a sequence of time-ordered GPS entries and represented by $Tr = \langle r_i, r_{i+1}, \ldots, r_{i+l-1} \rangle$. l is a user-specified

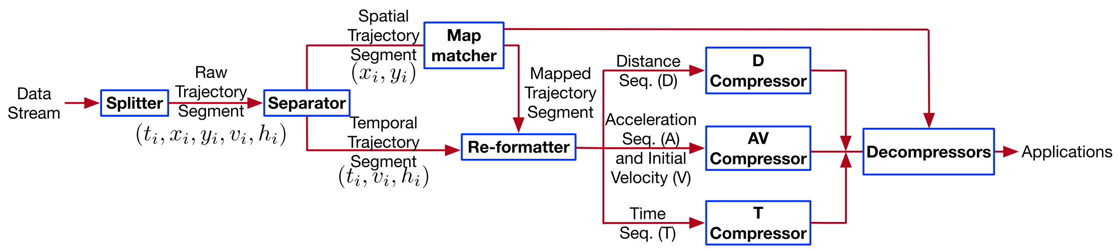


Fig. 2. The framework of DAVT system.

parameter, which controls the length of the raw trajectory segment.

Definition 4. (Mapped Trajectory Segment): Given a segment Tr, its mapped segment $\overline{Tr}$ is the spatial path that a vehicle truly travels in the road network. $\overline{Tr}$ is represented by a node sequence ($\langle n_1, n_2, \ldots, n_l \rangle$), in which two consecutive nodes ($\langle n_i, n_{i+1} \rangle$) are connected by a unique edge.

To obtain a corresponding mapped trajectory segment for a raw trajectory segment, one can turn to map-matching algorithms for help, such as ST- and SD-Matching algorithms [5], [22]. Two nice survey papers overview the trajectory data map-matching approaches from the perspectives of algorithms and applications, respectively [17], [25].

Definition 5 (Network Distance Between Two Points): The network distance between two points is defined as the length of the spatial path starting from one point to the other one in the road network. If GPS points do not seat on the edge, the network distance between two GPS points p_i and p_j is defined as the length of the path starting from their projection points p'_i to p'_j on the mapped trajectory segment. In the rest of the presentation, the network distance is called distance for short.

B. System Overview

We propose a system framework named DAVT to compress the raw trajectory segment in the temporal dimension (i.e., temporal trajectory), as shown in Fig. 2. In total, DAVT contains seven components, i.e., *Splitter*, *Separator*, *Map Matcher*, *Re-formatter*, and *D, AV and T Compressors*. As the GPS trajectory data is coming in stream, *Splitter* first divides the data stream into non-overlapped and equal-length raw trajectory segments, each of which is the basic data unit for data compression. *Separator* decomposes each segment into a pair of spatial and temporal trajectory segments. For the spatial segment, its mapped trajectory segment is obtained based on *Map matcher* in advance. Both the mapped trajectory and the temporal trajectory segments are input into *Re-formatter*, which would return a novel temporal trajectory representation. Such representation consists of three kinds of sequences, i.e., a distance sequence (D), an acceleration sequence and an initial velocity (AV), and a time sequence (T) of the segment (See Section IV). Next, the size of each element is reduced by *D*, *AV* and *T Compressors*, respectively (See Section V). Finally, the compact temporal trajectory segment is achieved. The data decompression (See Section V) should be applied before supporting applications, including real-time vehicle management and other offline Location-based Services (LBSs).

IV. A Novel Temporal Trajectory Representation

For a raw trajectory segment Tr, its novel temporal trajectory representation $\widetilde{Tr}$ is formed using the following DAVT format, represented by $\widetilde{Tr} = (D, AV, T)$.

D: *Distance Sequence for a Temporal Trajectory Segment.* The sequence $D(Tr)$ is proposed to record the exact positions of the vehicle. Different from storing the two-dimensional coordinates (i.e., latitude and longitude) directly, here we use the one-dimensional coordinate to represent the position. Specifically, along the mapped trajectory in the road network, we record the accumulated distance from the starting node of $\overline{Tr}$ to each point p_i to represent the actual position of the vehicle at ith sampling time. Such distance is referred to as L_i. Apparently, it is hard to compress the sequence of L_i since it contains little duplicate information. We use d_i to denote the distance between the two consecutive points p_{i-1} and p_i. d_1 is an exception, which refers to the distance from p_1 to the starting node of the spatial path. As a result, the equation $L_i = L_{i-1} + d_i = d_1 + \sum_{j=2}^{j=i} d_j$ holds mathematically. Therefore, we retain all d elements, so that the position of the vehicle at any sampling time (i.e., L_i) can be *recursively* inferred.

Since d_i varies significantly with the GPS sampling time interval based on our observations, we normalize it to a relative distance, denoted by $r(d_i)$, by dividing it to the square of the time interval, i.e., $(t_i - t_{i-1})^2$. Consequently, the proposed distance sequence is represented by $D(Tr) = \langle r(d_1), r(d_2), \ldots, r(d_l) \rangle$. Similarly, $r(d_1)$ equals to the ratio value of d_1 to the square of the sampling time interval. If distance sequence $D(Tr)$ is expected to contain some same distance values, we can just claim it is duplicate and can be further reduced by the encoding compression.

AV: *Acceleration Sequence and Velocity Sequence for a Temporal Trajectory Segment.* The sequences $A(Tr)$ and $V(Tr)$ are proposed to record another two dimensions of temporal information about the raw trajectory segment, i.e., average acceleration and instantaneous velocity, respectively. Specifically, $A(Tr)$ consists of a sequence of average acceleration $\overline{a_i}$ between two consecutive GPS points p_i and p_{i+1}, represented by $A(Tr) = \langle \overline{a_1}, \overline{a_2}, \ldots, \overline{a_{l-1}} \rangle$, where $\overline{a_i}$ is defined as the ratio of the velocity difference of two GPS points (i.e., p_i and p_{i+1}) to their time interval. $V(Tr)$ only stores the initial velocity of the raw trajectory segment (i.e., v_1), since other velocities can be deduced by associating the retained v_1, acceleration $\overline{a_i}$ and timestamps t_i, according to the following formula, as shown in Eq. (1).

$$v_i = \begin{cases} v_1 & , i = 1 \\ v_1 + \sum_{j=2}^{j=i} (t_j - t_{j-1}) \times \overline{a_{j-1}} & , i \geq 2 \end{cases} \quad (1)$$

T: *Time Sequence for a Temporal Trajectory Segment.* The sequence $T(Tr)$ is proposed to record timestamps of a raw trajectory segment, represented by $T(Tr) = \langle t_1, t_2, \ldots, t_l \rangle$. The representation of timestamps *facilitates the association* with the other two sequences, i.e., D and AV, since each element in them is *temporally constrained*. For example, d_i in sequence $D(Tr)$ is one-to-one correspondence with t_i in sequence $T(Tr)$. By combining $T(Tr)$ with other sequences, common LBS queries can be supported, which will be introduced in the next subsection.

In general, the proposed temporal trajectory segment using DAVT format can be represented by $\widetilde{Tr} = (\langle r(d_1), r(d_2), \ldots, r(d_l) \rangle, \langle \overline{a_1}, \overline{a_2}, \ldots, \overline{a_{l-1}} \rangle, v_1, \langle t_1, t_2, \ldots, t_l \rangle)$.

V. DAVT Compressor and Decompressor

The temporal trajectory segment consists of three sequences D, AV and T. Our target is to obtain concise and compact ones. Thus, we separately design three compressors (i.e., *D*, *AV* and *T Compressors*) to reduce the data size of each corresponding sort of sequence, respectively.

A. D Compressor

1) Procedure 1: Discretizing a Distance Sequence: A distance sequence is represented by $\langle r(d_1), r(d_2), \ldots, r(d_l) \rangle$. For each included element in the sequence, it could end in an infinite string in theory if GPS devices are sufficiently precise. To reduce the storage space, we adopt a straightforward method, i.e., discretization. For each element $(r(d_i))$, we obtain its corresponding discrete value $r(d_i)'$ based on Eq.(2), where δ represents the degree of dispersion and $i = 2\lfloor \frac{r(d_i)}{\delta} \rfloor$.

$$r(d_i)' = \begin{cases} \frac{i\times\delta}{2} & \text{if } |\mathrm{r(d_i)} - \frac{i\times\delta}{2}| < |\mathrm{r(d_i)} - \frac{(i+1)\times\delta}{2}| \\ \frac{(i+1)\times\delta}{2} & \text{if } |\mathrm{r(d_i)} - \frac{i\times\delta}{2}| \geq |\mathrm{r(d_i)} - \frac{(i+1)\times\delta}{2}| \end{cases} \quad (2)$$

Supposing a distance element $r(d_i)$ is 1.2345678 and δ is 0.5, its corresponding discrete value $r(d_i)'$ is 1.25. Except for economizing the memory space, such simple discretization can facilitate further compression, since the discrete distance sequence would be more likely to contain more duplicated elements. Meanwhile, the original distance sequence has been replaced by $D(\mathring{Tr}) = \langle r(d_1),' r(d_2),' \ldots, r(d_l)' \rangle$ and input for further processing. Note that the discretization will inevitably result in the loss of accuracy, indicating that the retained distance values in the compressed result will certainly deviate from the true ones. The bigger value of δ is, the more distance information $D(\mathring{Tr})$ will lose. To bound the loss, users can specify a suitable value of δ, according to the maximum error range that the concrete application can tolerate.

2) Procedure 2: Encoding a Distance Sequence via a Huffman Tree: We conduct an observation study based on a large sample of distance sequences and find that the included elements in each distance sequence have *few and limited* number of values ($\ll l$). In more detail, over 70% of all samples just have 4 unique values when δ and l are set to 1.5 and 10, respectively. Such an interesting observation makes the adoption of *Huffman encoding* a perfect choice to compress each element $r(d_i)'$ in a distance sequence $D(\mathring{Tr})$. The compressed element is denoted by $r(d_i)''$. The compressed result is denoted as $D(Tr) = \langle r(d_1),'' r(d_2),'' \ldots, r(d_l)'' \rangle$.

Algorithm 1: Huffman Tree Building.

```
Require:
    $D(\mathring{Tr}) = \langle r(d_1),' r(d_2)', \ldots, r(d_l)' \rangle$.
Ensure:
    A Huffman (binary) tree for $D(\mathring{Tr})$.
1:  Creating a set $R = \{r(d_i),' i = 1, \ldots, m\}$.
2:  Creating a set $F = \{f(r(d_i)'), i = 1, \ldots, m\}$.
3:  Creating a set $S = \{s_i | s_i.r = r(d_i)', s_i.f = f(r(d_i)'), i = 1, \ldots, m\}$.
4:  while $|S| != 1$ do
5:      Selecting two nodes $s_x$ and $s_y$ with minimum frequency.
6:      Creating a new parent node $s_z$ whose children nodes are $s_x$ and $s_y$
7:      $s_z.f = s_x.f + s_y.f$ and $s_z.r = Null$.
8:      Removing $s_x$ and $s_y$ from the set $S$.
9:      Adding $s_z$ in the set $S$.
10: end while
```

The main idea of Huffman encoding is to encode each element using the variable-length binary codes. The more frequent that the element appears in the sequence, the shorter binary code that is expected to be used to encode. Next, for an input distance sequence after discretization, we briefly present the procedure of Huffman encoding, which mainly consists of *Huffman tree building* (See Algorithm 1) and *element encoding* on top of the constructed tree.

Huffman Tree Building. Algorithm 1 summarizes the procedure of Huffman tree building. For an input distance sequence $D(\mathring{Tr})$, it outputs a Huffman (binary) tree for $D(\mathring{Tr})$. Lines 1∼3 is the initialization, which prepares materials for the tree building. Lines 4∼9 is a loop which demonstrates the procedure of the tree building. Specifically, we create a set (R) to store all unique numbers ($r(d_i)'$) in the input sequence $D(\mathring{Tr})$ (Line 1). We also create a set (F) to store the frequency ($f(r(d_i)')$) of each unique number in the set R (Line 2). Next, we create a set (S) to store all nodes of the Huffman tree (Line 3). The ith node in S has two attributes named $s_i.r$ and $s_i.f$, which represent the id and weight (i.e., frequency), with values of $r(d_i)'$ and $f(r(d_i)')$, respectively. After the initialization, the loop begins and it will end until when S contains only one node (Line 4). In more detail, we first select two nodes s_x and s_y having the smallest two frequencies from S (Line 5). Next, based on two children nodes s_x and s_y, we create their parent node referred to as s_z. What's more, the frequency of the new node s_z is the summarization of $s_x.f$ and $s_y.f$ (Line 7). Finally, we remove nodes s_x and s_y from S and add the node s_z in it (Lines 8∼9).

Supposing an example distance sequence $D(\mathring{Tr}) = \langle 6, 6, 6, 6, 6, 6, 5, 7, 7, 8 \rangle$, its set R and F are shown in the left part of Fig. 3. The Huffman tree of the distance sequence is also shown in the right part of Fig. 3. Here, in the tree, a circle refers to a node s_i and the number inside refers to its corresponding weight (i.e., $s_i.f$). The red numbers refer to the

R	F
5	0.1
6	0.6
7	0.2
8	0.1

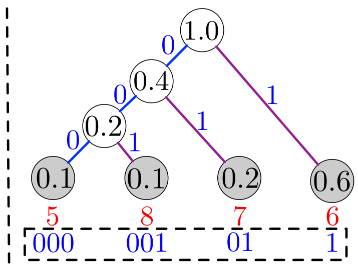


Fig. 3. An illustrative example of the Huffman tree building and element encoding. (Best viewed in the digital version.)

id (i.e., $s_i.r$) of these nodes. For simplicity, the $Null$ id is not shown in the tree.

Element Encoding. We encode each element in the sequence $D(\mathring{Tr})$ based on the constructed Huffman tree. Specifically, each unique number in $D(\mathring{Tr})$ is actually represented by the leaf node in the tree. For a leaf node, its binary code is just the codes that the path from the root to itself carries (i.e., the blue line from the root node to the leaf node). As a common convention, bit '0' represents the case of following the left link and bit '1' represents the case of following the right link. Back to the example sequence, the binary code of each unique number is illustrated in the dashed box in the right part of the figure. Consequently, the sequence $D(\mathring{Tr}) = \langle 6, 6, 6, 6, 6, 6, 5, 7, 7, 8\rangle$ is compressed into $D(Tr) = \langle 1111110000101001\rangle$. If each decimal number in $D(\mathring{Tr})$ is represented by four binary codes and occupies four bits when storing, the data size of the sample sequence can be reduced from 40 bits to 16 bits after encoding.

3) Procedure 3: Encoding all Distance Sequences via a Huffman Forest: There are two *simple and naive* methods that can be used to encode all distance sequences. The first is that we put all sequences together and build a uniform Huffman tree based on the combined long sequence. However, we can easily know that the tree would certainly contain a huge number of nodes and layers, since the long distance sequence is expected to have a much wider value range and also include many more unique numbers. The other one is that we can just build a Huffman tree for each distance sequence. Unfortunately, such a simple method will produce many redundant trees that are also a waste of computation and memory resources. The issue becomes extremely critical especially in the mobile environment. Therefore, both intuitive methods are not feasible in practice. To alleviate the potential issues, we propose an idea of building a *Huffman forest*, which expects to contain a limited number of Huffman trees. Each Huffman tree inside shall be responsible for encoding a fraction of all distance sequences. In the next, we will demonstrate that our proposed solution is essentially feasible.

According to the definition, the value of an element $r(d_i)'$ depends on the distance of two consecutive GPS points. It is well-known that the value of such distance exactly equals to the product of the moving speed of the vehicle and the sampling time interval of two consecutive GPS points. Since the sample time interval is usually constant, $r(d_i)'$ is indeed determined by the travel speed in essence. For a distance sequence with a suitable length, it is highly possible that the travel speed fluctuates within a small range. As a matter of fact, *this is also the reason why the Huffman tree built on top of a single distance sequence is simple.* Here, we make use of the road type information where the distance sequence was generated when travelling to cluster different distance sequences. *The rationale behind is that distance sequences generated when travelling through the same type of roads expect to present a close value distribution since the same road type has similar traffic rules, and the same speed limits.* For each cluster of the distance sequences, we combine them into a long distance sequence, based on which a new Huffman tree is built with the same procedure shown in Algorithm 1. As predicted, the built tree shall not contain a great number of leaf nodes. Finally, our Huffman forest shall have eight Huffman trees in total, because the road network contains eight common road types, i.e., 'motorway,' 'trunk,' 'primary,' 'secondary,' 'tertiary,' 'residential,' 'service' and 'unclassified'. It is easy to differentiate each Huffman tree in the forest since it is independently stored with a unique name.

On top of the built Huffman forest, given a newly arrival distance sequence, we first identify and retrieve a Huffman tree according to the road type that it traversed. Then, we follow the same procedure to the above-mentioned **Element Encoding** to fulfill the element encoding.

B. AV Compressor

1) Procedure 1: Discretizing an Acceleration Sequence: An acceleration sequence $A(Tr)$ is represented by $\langle\overline{a_1}, \overline{a_2}, \ldots, \overline{a_{l-1}}\rangle$. Similar to $r(d_i)$, each element $\overline{a_i}$ in the sequence may also end in an infinite string. We also discretize it and obtain its discrete value denoted by $\overline{a_i}'$ based on Eq. (3), where β represents the dispersion degree and $i = 2\lfloor\frac{\overline{a_i}}{\beta}\rfloor$. After discretization, $A(Tr)$ has already been replaced by $A(\mathring{Tr}) = \langle\overline{a_1}', \overline{a_2}', \ldots, \overline{a_{l-1}}'\rangle$. Similarly, the discretization will inevitably cause the loss of accuracy, and users could specify different β values according to different applications.

$$\overline{a_i}' = \begin{cases} \frac{i\times\beta}{2} & if\ |\overline{a_i} - \frac{i\times\beta}{2}| < |\overline{a_i} - \frac{(i+1)\times\beta}{2}| \\ \frac{(i+1)\times\beta}{2} & if\ |\overline{a_i} - \frac{i\times\beta}{2}| \geq |\overline{a_i} - \frac{(i+1)\times\beta}{2}| \end{cases} \qquad (3)$$

2) Procedure 2: Encoding an Acceleration Sequence via a Huffman Tree: We also conduct an observation study based on a large sample of acceleration sequences and get a similar phenomenon to the distance sequence. Thus, we also adopt Huffman encoding to compress an acceleration sequence $A(\mathring{Tr})$ and obtain a compressed result denoted by $A(Tr) = \langle\overline{a_1}'', \overline{a_2}'', \ldots, \overline{a_{l-1}}''\rangle$. The detailed procedure of Huffman encoding has already been explained in Section V-A2.

3) Procedure 3: Encoding all Acceleration Sequences via a Huffman Forest: With the same motivation to encode all distance sequences, we also propose the idea of building a Huffman forest to achieve all acceleration sequences encoding. *The core technique is thus converted to identify clusters of acceleration sequences with similar value distribution.*

It is well-recognized that values inside an acceleration sequence are highly correlated to velocity variances, which are mainly influenced by driving styles, traffic conditions, surrounding spatial context and, so on [8], [18]. Driving styles are highly subjective and vary from person to person, while it is expected that traffic conditions and spatial context along proximate roads may be highly similar [11]. Therefore, there are two potential solutions to cluster acceleration sequences, i.e., identifying acceleration sequences generated by users with similar driving styles and generated when travelling through roads lying closely in the spatial domain. Here, we mainly aggregate acceleration sequences based on the proximity of the traversing roads when considering that our collected trajectory data do not contain information about driving styles explicitly. More specifically, we first split the whole city into $N \times N$ equal-sized grid cells and simply cluster acceleration sequences generated when travelling roads in the same grid cell. As a result, we will have N^2 clusters in total, requiring a number of N^2 Huffman trees. In fact, the number of unique Huffman trees is much smaller than N^2. On one hand, there are some unreachable grid cells by driving (e.g., rivers, mountains, buildings). On the other hand, acceleration sequences generated in some different grid cells may just result in the same Huffman tree.

Velocity Compressor: The element (i.e., initial instantaneous velocity v_1 of the trajectory segment) in the velocity sequence is simply compressed by rounding itself.

C. T Compressor

1) Procedure 1: Transforming a Time Sequence: A time sequence $T(Tr)$ is represented by $\langle t_1, t_2 \cdots, t_l\rangle$. Each element t_i in the sequence is a huge UNIX number (e.g., 1545188400), while the time interval Δt_{i+1} between two consecutive timestamps (i.e., t_i and t_{i+1}) is usually smaller (e.g., 6). To save the memory space, we replace each timestamp t_i $(i \leq l-1)$ in the sequence with a time interval Δt_{i+1}. Consequently, $T(Tr)$ is transformed into $T(\mathring{T}r) = \langle t_1, \Delta t_2, \Delta t_3, \ldots, \Delta t_l\rangle = \langle t_1, \Delta T(Tr)\rangle$.

The new time sequence $T(\mathring{T}r)$ consists of two components, i.e., a time interval sequence $\Delta T(Tr)$ and an initial timestamp t_1. In the following, we will show the details on encoding a time interval sequence.

2) Procedure 2: Encoding a Time Interval Sequence: For some time interval sequences, all elements (Δt_i) in each sequence are just identical and equal to the sampling time of GPS points. For some time interval sequences, elements in each sequence *slightly* float up or down around the sampling time due to the delay or unavailable of GPS signals. We define a time interval sequence as *a high-quality one* or *a low-quality one* according to whether all included elements are identical. Afterwards, we respectively choose *Run-Length Encoding (RLE) algorithm* for high-quality sequences and the *Huffman encoding* for low-quality ones to encode, detailed as follows.

Using RLE to compress a high-quality time interval sequence. RLE algorithm is usually applied in compressing highly duplicative data. Its main idea is that the same value occurs in many consecutive elements in a sequence are stored as a single value Δt_i and its count $Num(\Delta t_i)$. For instance, given a sequence $\langle 6,6,6,6,6,6,6,6,6,6\rangle$, its compressed result would be $(6, 10)$, where 6 and 10 refer to Δt_i and $Num(\Delta t_i)$, respectively, occupying 8 bits only.

Using Huffman encoding to compress a low-quality time interval sequence. It is more cost-effective to encode a low-quality time interval sequence using the Huffman encoding. We simply use the example shown in Fig. 3 again to illustrate such outperformance. Given the sequence $\langle 6,6,6,6,6,6,5,7,7,8\rangle$, when still using the RLE algorithm, the compressed result is $\langle (6,6),(5,1),(7,2),(8,1)\rangle$, still occupying 32 bits. In contrast, if using Huffman encoding technique, the data size of the compressed result is 16 bits. To build a universal Huffman tree, here, we take a naive and simple idea, i.e., putting an adequate number of time interval sequences with low quality together to form a long sequence and train a single Huffman tree based on it. Unlike the cases discussed previously, we argue that such a simple idea is quite feasible, because the newly formed long sequence is still expected to have very *few and limited* values because the sampling time interval floats up or down slightly only.

To differentiate the compression methods in encoding a time interval sequence, we intentionally add a *method label* referred to as ml ahead of the compressed result. In more detail, ‘1’ refers to the case of compression using the RLE algorithm, while ‘0’ refers to the case of compression using the Huffman encoding. At last, the sequence $\Delta T(Tr)$ is represented by $(ml, \Delta T(Tr)')$.

3) Procedure 3: Encoding all Time Sequences: We take the same process to encode all time sequences. More specifically, when a new time sequence $T(Tr)$ is arriving, we first transform it into a new time sequence (See Procedure 1), which consists of two components, i.e., a time interval sequence $\Delta T(Tr)$ and an initial timestamp t_1. Then we encode the time interval sequence according to methods discussed in Procedure 2. After compression, a time sequence $T(Tr)$ is represented by $T(\mathring{T}r) = \langle t_1, (ml, \Delta T(Tr)')\rangle$.

D. DAVT Decompressor

Similar to data compression algorithms, the data decompression is done for all three kinds of sequences separately, but it is much easier. Here, we briefly overview the procedures on how to decompress them.

For a compressed distance sequence, we first obtain the related road type information using the corresponding spatial trajectory. The Huffman tree can thus be retrieved with the road type information. Finally, it is straightforward to decompress the distance sequence. Taking the compressed distance sequence $\langle 1111110000101001\rangle$ as the example again, with the Huffman tree shown in the right part of Fig. 3, we can easily decode the binary codes into the sequence of $\langle 6,6,6,6,6,6,5,7,7,8\rangle$. The decompression procedure of the acceleration and velocity sequence is very similar to that of the distance sequence. The only difference is the determination of the Huffman tree. In this case, the Huffman tree is chosen according to the location of the city grid, where the trajectory was generated.

To decompress a time sequence, it generally takes two steps. In the first step, based on the indicator of the encoding method (i.e., method label), we determine the decompression method. In the second step, we decompress the time sequence with different methods. If the indicator is '1,' then the data decompression should be done by decoding RLE codes; otherwise, the data decompression is done with the same procedure to that of the distance sequence. It should be noted that there is only one universal Huffman tree in this case. For instance, if the compressed time sequence is $1, 6, 5$, the decompression result should be $\langle 6, 6, 6, 6, 6\rangle$.

VI. Evaluation

A. Experiment Setup

1) Data Preparation: One road network dataset and a taxi GPS trajectory dataset from Beijing City, China, are used in the evaluation. The road network can be freely downloaded and extracted from OpenStreetMap.[2] In total, it contains 141,735 nodes and 157,479 edges. GPS trajectory data is generated by 595 taxis in one week (from 15th to 21st September 2015), which contains over 17,000,000 GPS points. The sampling time of GPS points is around 6 seconds. Due to the delay of GPS signals, the time intervals of about 20% consecutive timestamps float up and down (i.e., 5 $\sim$ 12 seconds). In terms of storage space, the raw trajectory data takes up over 1.01 GB.

2) Baselines: Three compressors are used as baselines, detailed as follows.

PRESS. A segment of temporal trajectory in PRESS is represented by a sequence of tuples (t_i, d_i), where t_i and d_i refer to the sampling time and the accumulated travel distance on the spatial path since the engine starting time, respectively. A sequence can thus be viewed as a time-distance curve in a 2D plane. On this basis, similar to the idea of Douglas-Peucker (DP) [10], the data compression is achieved by only keeping the "feature" points in the plane. Readers can refer to [30] for more details.

CCF. It has the same temporal trajectory representation to PRESS. The only difference is that CCF tries to fit an optimal line to approximate the origin curve based on the idea of linear-fitting within the user-defined error bound. In other words, the fitted line generates a set of totally new points (except for the first point) and thus cause extra storage cost. Readers can refer to [12] for more details.

TED. The temporal trajectory is represented by two kinds of sequences, i.e., $r(p_i)$ and t_i, respectively, where $r(p_i)$ and t_i refer to the relative distance from p_i to the edge's head node that it located and the sampling time. For the distance sequence, TED takes the idea of encoding to compress data. The bigger value of $r(p_i)$ is, the more bits are used to encode. For the time sequence, the data reduction is achieved by discarding time sequences with the identical time interval. Readers can refer to [34] for more details.

Remark. All of the three baselines do not consider the compression of time-varying velocity information recorded in the trajectory data. To make the comparison fair, in the comparison study, we only show the performance results of *DT Compressor*.

3) Evaluation Metrics: We use the following metrics to evaluate the performance.

[2]www.openstreetmap.org

Compression ratio. The compression ratio (cr) that is the same to other data compressors is defined as the ratio of the occupied storage space of the raw data to that of the compressed data [30]. In general, cr is larger than 1.

Compression and decompression time. The time cost in the data compression and decompression respectively are used to measure the efficiency of the system performance [34].

Time synchronized network distance. For a distance sequence, time synchronized network distance ($TSND$) measures the distance loss caused by data compression [34]. Specifically, we simply use the maximum distance error to quantify. $TSND$ is defined in Eq.(4).

$$TSND = Max_{i=1}^{l}(|L_i' - L_i|) \tag{4}$$

where $|L_i' - L_i|$ is the distance error at a sampling time t_i referring to the difference between the distance computed based on the distance sequence before and after compression, denoted by L_i and L_i' respectively. In more detail, L_i refers to the accumulated distance value from the starting point along the spatial path at time t_i computed based on the raw distance sequence (before value discretization and encoding); L_i' refers to such accumulated distance value at time t_i computed based on the compressed distance sequence (after value discretization and encoding). Parameter l refers to the length of the trajectory segment. It is easy to understand that $TSND$ is tightly related to δ and l.

Time synchronized acceleration difference. Similarly, for an acceleration sequence, time synchronized acceleration difference ($TSAD$) measures the acceleration loss caused by data compression. Specifically, we use the maximum acceleration error to quantify. $TSAD$ is defined in Eq.(5).

$$TSAD = Max_{i=1}^{l}(|acc_i' - acc_i|) \tag{5}$$

where $|acc_i' - acc_i|$ is the acceleration error at a sampling time t_i referring to the difference between the acceleration computed based on the acceleration sequence before and after compression, denoted by acc_i and acc_i' respectively. In more detail, acc_i refers to the acceleration value at time t_i computed based on the raw acceleration sequence (before value discretization and encoding); acc_i' refers to such acceleration value at time t_i based on the compressed acceleration sequence (after value discretization and encoding). It is also easy to know that $TSAD$ is tightly related to β and l.

4) Evaluation Environment: All the evaluations in the paper are run in MATLAB on an Intel (R) Xeon (R) CPU E3-1275 PC with 32-GB RAM and Windows 10 operating system.

B. DAVT Compressor Training

An important step ensures that our proposed framework can work is the offline training, i.e., Huffman tree and forest building. Thus, we evaluate the training time, the storage space used to save Huffman models, and the compression ratio w.r.t the size of data used for training. Using less training data can benefit the maintenance of the framework and make it more competitive. Here, for simplicity, the size of the training data is measured by the time duration that the trajectory data were generated. The

TABLE I
# OF GPS POINTS VS TIME DURATION (IN DAY)

Time	0.1	0.3	0.5	0.7	0.9	1	2
#GPS	261K	784K	1302K	1822K	2351K	2439K	5177K

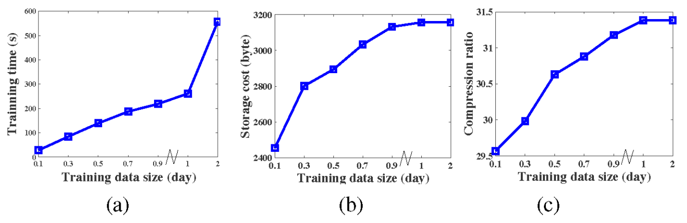


Fig. 4. Training performance under different data sizes.

TABLE II
TRAINING PERFORMANCE UNDER DIFFERENT NS

N	1	4	9	16	25
Training time (s)	231	240	262	269	285
Storage cost (byte)	220	900	3157	7700	16600
Compression ratio	31.0	31.4	31.7	32.1	32.8

relationship between the time duration (in a day unit) and the number of GPS points generated can be found in Table I.

We show the results of training performance under different training data sizes in Fig. 4. As can be seen in Fig. 4(a), as predicted, the training time climbs almost linearly as the training data gets bigger. As for the compression performance shown in Fig. 4(b) and (c), before reaching the steady state, both the space occupied by offline trained model and compression ratio increase gradually with the training data size. This is because that more unique Huffman trees would be identified when the training data gets larger at first, but no more new trees would be generated after the size of the training data reaches a certain threshold (i.e., one day). By combining training time and compression results shown in Fig. 4, we train our model using one-day trajectory data since it is able to trade-off the training time cost and the compression performance. In this experiment, we fix $\delta = 0.5, \beta = 0.5, l = 10$ and $N = 9$.

Another key parameter in the model training is the number of equal-sized city grids (i.e., N) in building the Huffman forest for *AV Compressor*. We test the training performance under different Ns,[3] with the results shown in Table II. As can be seen, slightly more training time costs since more training rounds are needed with more city grid cells. With the number of city grids, the storage cost of the trained model increases exponentially, while the improvement on the compression ratio is quite limited. To sum up, we fix $N = 9$ in the rest of experiments since it strikes a nice trade-off between the training costs and the improvement. In this experiment, we fix $\delta = 0.5, \beta = 0.5$, and $l = 10$.

C. Information Loss

We are also interested at the information loss in real cases. Thus, here we investigate the information loss of distance and

[3]Here, we use the symbol of ·s to denote different choices of ·, e.g., Ns refer to different values of N.

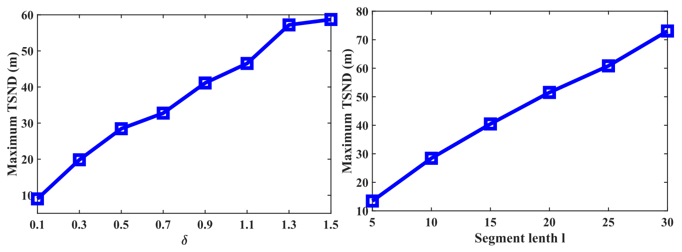


Fig. 5. Distance loss under different δs (left) and ls (right).

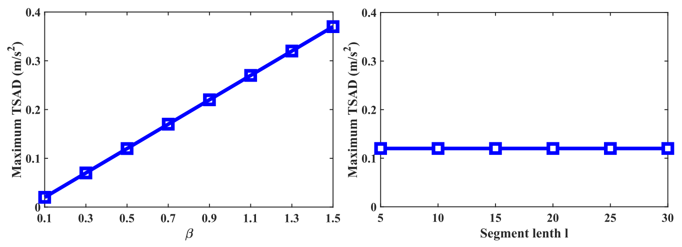


Fig. 6. Acceleration loss under different βs (left) and ls (right).

acceleration under different parameter settings in the testing data.

1) Loss of Distance Information: As discussed, we use the $TSND$ to evaluate the distance loss for a distance sequence. For a set of distance sequences, we simply use the maximal $TSND$ among all sequences to quantify the overall distance information loss. We argue that the distance loss in the worst case is more useful in practice. If the maximum distance loss is acceptable, so is the lossy compression framework.

Fig. 5 shows the results of maximum distance loss under different δs and ls. As expected, caused by the data discretization, the distance information loss becomes larger as δ gets bigger. The maximum distance error is no more than 60 meters, even if we set δ to 1.5, indicating that the *D Compressor* is quite accurate. As can be seen from the right part of Fig. 5, the maximum $TSND$ increases linearly with the length of the segment. This is because that the distance error accumulation effect becomes more serious as the segment grows longer (see the definition of $TSND$ for details).

2) Loss of Acceleration Information: With the same motivation, we also use the maximum of $TSAD$ among all acceleration sequences to quantify the overall acceleration loss. We investigate the maximum $TSAD$ under different βs and ls, as shown in Fig. 6. As can be predicted, similar to the case of distance loss, more information would be lost if β gets bigger. As shown in the right part of Fig. 6, the maximum $TSAD$ remains unchanged under different ls. This is because $TSAD$ is defined as the velocity variance between two consecutive sampling times, which is not an accumulated value of l as $TSND$.

D. DT Compressor vs Baseline Algorithms

All three baselines are lossy compression algorithms. In general, they can only work after setting the predefined error range (i.e., distance). Based on the results shown in the study of distance information loss (Fig. 5), it is also not difficult to pick

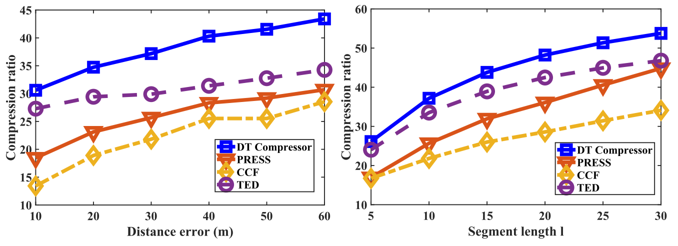


Fig. 7. Comparison results on the compression ratio under different distance errors (left) and ls (right).

up a suitable parameter setting (i.e., δ), which is expected to result in a similar error range. For instance, we can set $\delta = 0.5$ if we expect the distance error of fewer than 30 meters. Via this simple mapping, we are able to compare the compression performance of *DT Compressor* to the other three baselines in terms of compression ratio, compression and decompression time cost, with results shown as follows.

1) Compression Ratio: Fig. 7 shows the comparison results on the compression ratio under different distance errors and ls, respectively. From the left part of Fig. 7, as can be observed, for all four compression algorithms, the compression ratio increases steadily with the distance error. It is easy to understand that the compression performance is better if we allow a bigger loss of distance information. Moreover, the proposed *DT Compressor* achieves a consistently higher compression ratio than the three baselines, demonstrating the effectiveness of the new representation and compression algorithm.

The right part of Fig. 7 shows the comparison results under different lengths of the trajectory segment. For all four algorithms, the compression ratio becomes bigger if l gets larger. Again, the proposed *DT Compressor* outperforms consistently. For *DT Compressor* and TED, both contain two parts (i.e., distance and time) in the compressed data. As for the distance part, the size of compressed data is determined by the binary codes that are used to encode each element and the number of total elements. No matter what the value of l is, for a fixed testing dataset, the element representations and the number of the total elements are both unchanged. Thus, its size shall be *unchanged* and independent with regard to the length of the segment. In contrast, as for the time part, the number of segments regarding time shrinks when l increases, leading to a *reduction* of space occupation. It should be noted that, for each segment regarding time, its size is usually constant after data compression. Therefore, the compressed data occupies less space if the trajectory segment gets longer.

2) Compression and Decompression Time Cost: Fig. 8 shows the comparison results on the compression and decompression time cost under different distance errors and ls, respectively. As can be seen from Fig. 8(a) and (b), for *DT Compressor* and TED, less time is needed in data compression if allowing a bigger distance error. At the same time, less time is required in data decompression. For *DT Compressor*, a bigger distance error indicates a relatively bigger value of δ in data discretization. In this case, a simpler Huffman tree structure and a smaller number of trees are adequate to encode all data

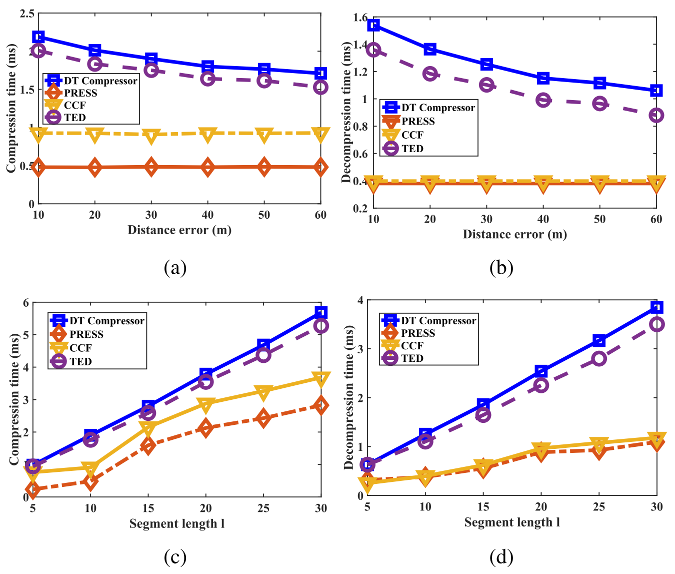


Fig. 8. Comparison results on the compression and decompression time cost under different distance errors and ls.

elements, costing less compression and decompression time. For the TED algorithm, a bigger distance error also indicates a more coarse resolution in data discretization, leading to a fewer number of unique values. Consequently, both data compression and decompression become much easier and more efficient. For PRESS and CCF algorithms, according to their working principles, the time cost is independent with respect to the distance error. For instance, given any distance error, PRESS should always determine whether to retain or discard the current data point. Therefore, they cost stable time in data compression and decompression across all distance errors.

With respect to the length of the trajectory segment, as can be seen from Fig. 8(c) and (d), for all four algorithms, they cost more time in data compression and decompression as l gets bigger. This observation is easy to understand since more time is required to handle more data elements (points) if setting a bigger l. *DT Compressor* costs consistently more time than the other three baselines. We can also draw the conclusion that *a better performance in the compression ratio (i.e., effectiveness) is often at the cost of the increase of compression and decompression time (i.e., efficiency)*. But it should be noted that our proposed *DT Compressor* is still efficient enough, since the biggest time cost (i.e., $l = 30$) is less than 6 millisecond, which is small enough.

In summary, for all four algorithms, data decompression is more efficient than the data compression, and they can be done in milliseconds. In terms of time cost, *DT Compressor* performs slightly worse than the TED algorithm in both data compression and decompression. This is because that we take a more sophisticated and accurate time compressor to deal with the time part. Meanwhile, it also costs more time in data decompression. As a comparison, both PRESS and CCF perform much more efficiently in data compression and decompression.

3) Trade off Between Compression Ratio and Compression/Decompression Time: Combining the left part of Fig. 7 and

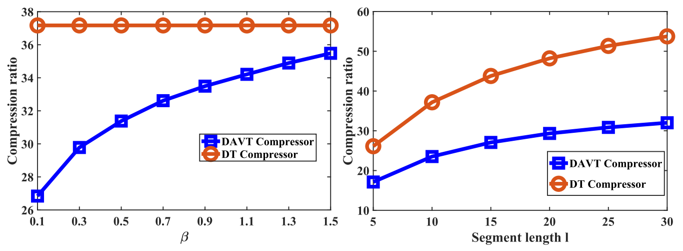


Fig. 9. Comparison results between *DT* and *DAVT Compressor* on the compression ratio under different βs and ls.

Fig. 8(a) and (b) together, we can easily know the compression ratio, compression and decompression time under different distance errors. As can be observed, for *DT Compressor* and TED, with increasing distance errors, the compression performance becomes better with both higher compression ratio and less time cost for compression and decompression. Slightly differently, for the PRESS and CCF algorithms, if we allow a bigger distance error, only compression ratio becomes higher, while the compression and decompression time remain almost unchanged. Based on the analysis results, For all four algorithms, there is no need to worry about the trade off between the compression ratio and compression/decompression time cost under different distance errors. It is safe to choose the maximal distance error only if it satisfies the requirement of applications (i.e., the user-defined error bound).

Similarly, we can obtain the compression ratio, compression and decompression time under different ls after combining the right part of Fig. 7 and Fig. 8(c) and (d) together. For all four algorithms, although a long segment length can improve the compression ratio, it will inevitably consume more time cost for both compression and decompression. Thus, it is necessary to select an approximate length l to balance the compression ratio and compression/decompression time cost when implementing compressors in the real applications.

E. DT Compressor vs DAVT Compressor

It can be expected that the performance would degrade in both effectiveness and efficiency since *DAVT Compressor* takes into account the time-varying velocity information, instead of simply discarding it. To quantify the performance degradation, we measure the differences in terms of compression ratio, and compression and decompression time cost between *DT Compressor* and *DAVT Compressor*.

1) Compression Ratio: Fig. 9 shows the comparison results on the compression ratio between *DT* and *DAVT Compressor*. Fig. 9(a) shows that the compression ratio of *DT Compressor* under different βs keeps unchanged since it is not dependent on β. For *DAVT Compressor*, a bigger β indicates a coarser resolution in data discretization and more information loss, leading to an improvement in compression ratio. The reason is exactly the same to the case of the compression ratios under different δs. Furthermore, we also observe that the difference of compression ratios between the two compressors abridges when β increases.

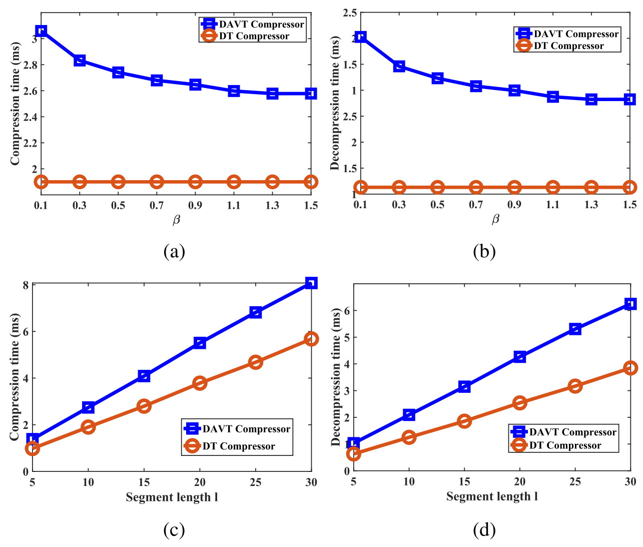


Fig. 10. Comparison results between *DT* and *DVAT Compressors* in terms of compression and decompression time cost under different βs and ls.

As shown in Fig. 9(b), both *DT* and *DAVT Compressors* achieve a higher compression ratio with a bigger l. For *DAVT Compressor*, it considers two more parts (i.e., acceleration and velocity) than *DT Compressor*. Similar to the distance part, the space occupied by the acceleration part after data compression is not dependent on segment length. Fewer velocity and time parts would be generated with a bigger l, leading to an increase in the compression ratio eventually.

2) Compression and Decompression Time Cost: Fig. 10 shows the comparison results on the compression and decompression time cost under different βs and ls. It is easy to understand the compression and decompression time of *DT Compressor* should be the same under different βs since they are independent, as shown in Fig. 10(a) and (b). For *DAVT Compressor*, both compression and decompression time decline as β gets bigger. The reason is also exactly the same to the study of *DT Compressor* w.r.t. δ. Both compressors are quite efficient and respond in milliseconds.

Fig. 10(c) and (d) show the comparison results between the two compressors. For both *DT* and *DAVT Compressors*, their. The reason is that both compressors take the same time to compress each data element (point), and *DAVT* takes relatively more time to compress additional velocity information. We can also observe that the data decompression usually costs less time than data compression.

VII. Conclusion and Future Work

In this paper, we present a novel trajectory data representation and compression framework, with the objective lowering the data communication and storage costs at the same time by compressing data at the side of data generators. Meanwhile, we also considered the time-varying information carefully and devised a well-performed compressor to deal with it. Extensive experimental results demonstrated its superior performance in both compression effectiveness and efficiency.

In the future, we plan to broaden and deepen this work in the following directions. First, we intend to investigate the time effect (e.g., different times of the day, different days of the week) in the system performance during compressing distance, acceleration, and velocity sequences. Second, we only consider the spatial proximity in building the Huffman forest in designing *AV Compressors* at the current stage. In the near future, we plan to explore the usability of driving styles of different drivers and check whether it improves and to what degree. Finally, we plan to transplant our compression algorithms into the current GPS devices and test the real performance in the field study.

References

[1] H. Cao, O. Wolfson, and G. Trajcevski, "Spatio-temporal data reduction with deterministic error bounds," *The VLDB J.—The Int. J. Very Large Data Bases*, vol. 15, no. 3, pp. 211–228, 2006.

[2] J. Capon, "A probabilistic model for run-length coding of pictures," *IRE Trans. Inform. Theory*, vol. 5, no. 4, pp. 157–163, 1959.

[3] P. S. Castro, D. Zhang, C. Chen, S. Li, and G. Pan, "From taxi GPS traces to social and community dynamics: A survey," *ACM Comput. Surv. (CSUR)*, vol. 46, no. 2, p. 17, 2013.

[4] C. Chen, Y. Ding, Z. Wang, J. Zhao, B. Guo, and D. Zhang, "VTracer: When online vehicle trajectory compression meets mobile edge computing," *IEEE Syst. J.*, vol. 14, no. 2, pp. 1635–1646, Jun. 2020.

[5] C. Chen, Y. Ding, X. Xie, and S. Zhang, "A three-stage online map-matching algorithm by fully using vehicle heading direction," *J. Ambient Intell. Humanized Comput.*, vol. 9, no. 5, pp. 1623–1633, 2018.

[6] C. Chen, Y. Ding, X. Xie, S. Zhang, Z. Wang, and L. Feng, "TrajCompressor: An online map-matching-based trajectory compression framework leveraging vehicle heading direction and change," *IEEE Trans. Intell. Transp. Syst.*, vol. 21, no. 5, pp. 2012–2028, May 2020.

[7] C. Chen, S. Jiao, S. Zhang, W. Liu, L. Feng, and Y. Wang, "Tripimputor: Real-time imputing taxi trip purpose leveraging multi-sourced urban data," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 10, pp. 3292–3304, 2018.

[8] Y. Ding, C. Chen, S. Zhang, B. Guo, Z. Yu, and Y. Wang, "GreenPlanner: planning personalized fuel-efficient driving routes using multi-sourced urban data," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. (PerCom)*, pp. 207–216, 2017.

[9] J.-X. Dong, C. Hicks, and D. Li, "A heuristics based global navigation satellite system data reduction algorithm integrated with map-matching," *Ann. Oper. Res.*, pp. 1–16, 2019.

[10] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *The Int. J. Geographic Inform. Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[11] R. Ganti, N. Pham, H. Ahmadi, S. Nangia, and T. Abdelzaher, "GreenGPS: a participatory sensing fuel-efficient maps application," in *Proc. ACM Int. Conf. Mobile systems, Appl., and Serv.*, pp. 151–164, 2010.

[12] Y. Ji, Y. Zang, W. Luo, X. Zhou, Y. Ding, and L. M. Ni, "Clockwise compression for trajectory data under road network constraints," in *Proc. IEEE Int. Conf. Big Data*, pp. 472–481, 2016.

[13] G. Kellaris, N. Pelekis, and Y. Theodoridis, "Trajectory compression under network constraints," *Adv. Spatial Temporal Databases*, pp. 392–398, 2009.

[14] T. Kieu, B. Yang, C. Guo, and C. S. Jensen, "Distinguishing trajectories from different drivers using incompletely labeled trajectories," in *Proc. 27th ACM Int. Conf. Inform. Knowledge Manag.*, pp. 863–872, 2018.

[15] D. E. Knuth, "Dynamic Huffman coding," *J. Algorithms*, vol. 6, no. 2, pp. 163–180, 1985.

[16] X. Kong, F. Xia, Z. Ning, A. Rahim, Y. Cai, Z. Gao, and J. Ma, "Mobility dataset generation for vehicular social networks based on floating car data," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 3874–3886, 2018.

[17] M. Kubicka, A. Cela, H. Mounier, and S.-I. Niculescu, "Comparative study and application-oriented classification of vehicular map-matching methods," *IEEE Intell. Transp. Syst. Mag.*, vol. 10, no. 2, pp. 150–166, 2018.

[18] C. F. Lee and P. Öberg, "Classification of road type and driving style using obd data, SAE Technical Paper 2015-01-0979, 2015, https://doi.org/10.4271/2015-01-0979.

[19] X. Lin, S. Ma, H. Zhang, T. Wo, and J. Huai, "One-pass error bounded trajectory simplification," *Proc. VLDB Endowment*, vol. 10, no. 7, pp. 841–852, 2017.

[20] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, and R. Jurdak, "Bounded quadrant system: Error-bounded trajectory compression on the go," in *IEEE Int. Conf. Data Eng. (ICDE)*, 2015, pp. 987–998.

[21] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, J.-G. Lee, and R. Jurdak, "A novel framework for online amnesic trajectory compression in resource-constrained environments," *IEEE Trans. Knowledge Data Eng.*, vol. 28, no. 11, pp. 2827–2841, 2016.

[22] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang, "Map-matching for low-sampling-rate GPS trajectories." in *Proc. 17th ACM SIGSPATIAL Int. Conf. Adv. Geographic Inform. Syst.*, 2009, pp. 352–361.

[23] Y. Lyu, V. C. Lee, J. K.-Y. Ng, B. Y. Lim, K. Liu, and C. Chen, "Flexi-Sharing: A flexible and personalized taxi-sharing system," *IEEE Trans. Veh. Technol.*, vol. 68, no. 10, pp. 9399–9413, Oct. 2019.

[24] N. Meratnia and A. Rolf, "Spatiotemporal compression techniques for moving point objects," in *Proc. Int. Conf. Extending Database Technol.*, 2004, pp. 765–782.

[25] M. A. Quddus, W. Y. Ochieng, and R. B. Noland, "Current map-matching algorithms for transport applications: State-of-the Art and future research directions," *Transp. Res. Part C: Emerging Technol.*, vol. 15, no. 5, pp. 312–328, 2007.

[26] R. Rana, M. Yang, T. Wark, C. T. Chou, and W. Hu, "Simpletrack: Adaptive trajectory compression with deterministic projection matrix for mobile sensor networks," *IEEE Sen. J.*, vol. 15, no. 1, pp. 365–373, Jan. 2015.

[27] K.-F. Richter, F. Schmid, and P. Laube, "Semantic trajectory compression: Representing urban movement in a nutshell," *J. Spat. Inform. Sci.*, vol. 2012, no. 4, pp. 3–30, 2012.

[28] F. Schmid, K.-F. Richter, and P. Laube, "Semantic trajectory compression," in *Proc. Int. Symp. Spatial Temporal Databases*, 2009, pp. 411–416.

[29] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Int. Things J.*, vol. 3, no. 5, pp. 637–646, 2016.

[30] R. Song, W. Sun, B. Zheng, and Y. Zheng, "PRESS: A novel framework of trajectory compression in road networks," *Proc. VLDB Endowment*, vol. 7, no. 9, pp. 661–672, 2014.

[31] J. V. Stone," *Inform. Theory: A Tutorial Introduction*, "Sebtel Press," 2015.

[32] J. Uthayakumar, T. Vengattaraman, and P. Dhavachelvan, "A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications," *J. King Saud University-Comput. Inform. Sci.*, 2018, doi: 10.1016/J.JKSUCI.2018.05.006.

[33] J. Wang, K. Liu, K. Xiao, X. Wang, Q. Han, and V. C. S. Lee, "Delay-constrained routing via heterogeneous vehicular communications in software defined busnet," *IEEE Trans. Veh. Technol.*, vol. 68, no. 6, pp. 5957–5970, Jun. 2019.

[34] X. Yang, B. Wang, K. Yang, C. Liu, and B. Zheng, "A novel representation and compression for queries on trajectories in road networks," *IEEE Trans. Knowl. Data Eng.*, vol. 30, no. 4, pp. 613–629, Apr. 2018.

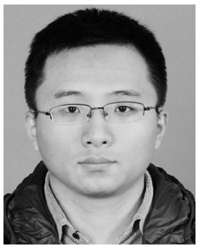
[35] D. Zhao and E. Stefanakis, "Integrated compression of vehicle spatio-temporal trajectories under the road stroke network constraint," *Trans. GIS*, vol. 22, no. 4, pp. 991–1007, 2018.

[36] Y. Zhao, S. Shang, Y. Wang, B. Zheng, Q. V. H. Nguyen, and K. Zheng, "REST: A reference-based framework for spatio-temporal trajectory compression," in *Proc. 24th ACM SIGKDD Int. Conf. Knowl. Discovery & Data Mining*, pp. 2797–2806, 2018.

[37] K. Zheng, Y. Zhao, D. Lian, B. Zheng, G. Liu, and X. Zhou, "Reference-based framework for spatio-temporal trajectory compression and query processing," *IEEE Trans. Knowl. Data Eng.*, vol. 32, no. 11, pp. 2227–2240, Nov. 2020.

[38] J. Ziv and A. Lempel, "A universal algorithm for sequential data compression," *IEEE Trans. Inform. Theory*, vol. 23, no. 3, pp. 337–343, 1977.

Chao Chen received the B.Sc. and M.Sc. degrees in control science and control engineering from Northwestern Polytechnical University, Xi'an, China, in 2007 and 2010, respectively, and the Ph.D. degree from Pierre and Marie Curie University and Institut Mines-Télécom/Télécom SudParis, France, in 2014. He is currently a Full Professor with the College of Computer Science, Chongqing University, Chongqing, China. His research interests include data mining from large-scale GPS trajectory data, and big data analytics for smart cities.



Yan Ding received the undergraduate and master's degrees from Chongqing University, Chongqing, China, in 2016 and 2019, respectively. He has been working toward the Ph.D. degree majoring in computer science with State University of New York at Binghamton, Binghamton, NY, USA, since the fall semester 2019. His research interests are Task and Motion Planning (TMP): the intersection of Planning from Artificial Intelligence and Motion Planning from Robotics (including vehicles).

Suiming Guo is currently an Associate Professor with the College of Information Science and Technology and College of Cyber Security, Jinan University, Guangzhou, China. He is interested in the Internet-related industry, as well as the academia. His research interests include big data mining, analysis, modelling, city-level data mining and smart cities.

Yasha Wang is currently a Professor and the Associate Director of National Research and Engineering Center of Software Engineering, Peking University, Beijing, China. His research interests include urban data analytics, ubiquitous computing, and collaborative software development.