# Chapter 2
# Trajectory Data Compression


## 2.1 Introduction

Due to the increasing of GPS-equipped public and private vehicles, trajectory data is accumulated and recorded rapidly, massively and abundantly. Trajectory data is sending to the data center continuously, causing communication overhead, storing and computing issues [1–3]. To make matter worse, massive data also makes computing tasks (e.g. data visualization, pattern mining) more complicated, resulting in worse performance. To ameliorate the above-mentioned issues, reducing the size of trajectory data before sending it to the data center is one of promising ways [4]. Intuitively, the easiest way is collecting the vehicle's locations less frequently. However, such method is problematic because the collected data may be too sparse to infer the detailed driving paths in-between, resulting in limited urban services [5]. On the contrary, trajectory compression based on online map-matching is a common but never well-addressed solution, which tries to address the issues from the perspective of seeking trajectory representation [6]. To be more specific, map-matching algorithm aligns raw trajectory onto the road network in advance. The mapped trajectory returned in terms of a sequence of connected edges in the road network is usually redundant since some edges in the trajectory can be recovered using the reserved edges. The objective of trajectory compression is to reduce such redundancy. However, there is an obvious conflict between the resource-hungry map-matching and trajectory-compressing tasks and the

Part of this chapter is based on two previous work:

C. Chen, Y. Ding, S. Zhang, Z. Wang and L. Feng, "TrajCompressor: An online map-matching-based trajectory compression framework leveraging vehicle heading direction and change," IEEE Transactions on Intelligent Transportation Systems, vol. 21, no. 5, pp. 2012–2028, Apr. 2019.

C. Chen, Y. Ding, Z. Wang, J. Zhao, B. Guo and D. Zhang, "VTracer: When online vehicle trajectory compression meets mobile edge computing," IEEE Systems Journal, vol. 14, no. 2, pp. 1635–1646, Aug. 2019.



C. Chen et al., *Enabling Smart Urban Services with GPS Trajectory Data*,
https://doi.org/10.1007/978-981-16-0178-1_2

resource-limited vehicle-mounted GPS devices. The vehicle-mounted GPS devices cannot undertake the heavy tasks themselves.

In this chapter, similar to the SD-Matching algorithm discussed in Chap. 1, we also intend to explore the heading direction information in online trajectory compression. To be more specific, we propose a heading-change-compression (HCC) algorithm, which only maintains edges whose heading change significantly, compared to the previously connected edges. To alleviate the issue caused by the limited computing capacity of GPS devices, we propose an idea of leveraging the smartphones of drivers as the local computing unit (i.e., cloudlet) to migrate the computing burdens, as inspired by the mobile edge computing [7]. In summary, the main contributions of this chapter are as follows.

- Based on the mapped trajectory, we propose a novel online trajectory compression algorithm that leverage the heading direction in a smart way. We also conduct extensive evaluations using real-life trajectory data generated by 595 taxis in a week and road network data in the city of Beijing to verify the efficiency and effectiveness of HCC. Experimental results show that HCC achieves a high compression ratio and can respond in real time.
- We develop a system called VTracer combining HCC with SD-Matching, which can save the cost on storage, communication significantly. The main noteworthy aspect of VTracer is that it brings the idea of mobile edge computing into a very preliminary but important step of trajectory data mining tasks (in the step of data collection). With VTracer, the data center can obtain a noise-free, more concise yet still completed trajectory representation for the moving vehicles. We also evaluate VTracer system in terms of the trajectory mapping quality, the compression ratio, memory, and energy consumptions, and the app size in real situations, using the 13-h real driving data generated in the city of Chongqing, China (equivalent to around 530 km in driving distance). Experimental results demonstrate the superior performance of VTracer.

## 2.2 Related Work

According to the utilization of the road network, trajectory compression algorithms are generally classified into two groups, i.e., line-simplification-based and map matching-based algorithms.

### *2.2.1 Line-Simplification-Based Compression*

The GPS points of the raw trajectory can be divided into important and unimportant points in terms of their contribution to the trajectory shape. And the core idea of line-simplification-based algorithms is to maintain some important GPS points [8]. To

evaluate the importance for a given GPS point, the distance from the point to the line segment composed by the end-points of the (sub-)trajectory is calculated. The GPS point is claimed important if the distance is bigger than the user-specified error bound. Therefore, the error bound value has a significant impact on the number of retained GPS points and the compression ratio. The DP and its variants are the most well-known representatives [8]. The importance of the GPS point is counted by considering the GPS location only, thus it can be wrongly evaluated due to the measurement errors. To alleviate such side-effect, map-matching-based algorithms are proposed by aligning the GPS points to the underlying road network before trajectory compression.

### *2.2.2 Map-Matching-Based Compression*

The core idea of map-matching-based compression is mapping raw trajectory data onto the road network in advance [9, 10]. The utilization of road network further benefits the trajectory compression since it endows the trajectory with more correct and semantical information. Some recent work also introduces the trajectory prediction in the compression. Based on the historical trajectory data [11], will retain the trajectory which is not consistent with the results predicted by the Markov model, and discard that is consistent. In the overview of trajectory data mining [12], briefly summarize some popular trajectory compression algorithms. Here, we select three well-known compressors to review, i.e., MMTC [13], PRESS [3], CCF [1].

We know that in a road network there can be multiple paths between two points, and each path usually consists of a different number of edges, meaning that some paths just have more edges than others. In order to reduce the number of edges, MMTC algorithm makes use of the paths with fewer edges but with high similarity to replace the original sub-trajectory. In real life, most drivers prefer to select the shortest paths. Thus, PRESS utilizes the driving preference to accomplish trajectory compression. More specifically, PRESS replaces the shortest sub-path in the original trajectory by the pair of beginning and ending edges. However, the shortest path computation either causes high computation time (online) or storage space cost (offline), which greatly degrades the efficiency. Different from PRESS, CCF explores the usability of intersections in trajectory compression. It retains all out-edges at intersections to represent the compressed trajectory. Since the number of out-edges is much smaller than the edges in the mapped trajectory, CCF can save significant storage space. However, retaining all out-edges is still redundant and the trajectory can be further compressed. Going a step further, in this study, we propose a novel trajectory compression algorithm, which takes full advantage of heading changes at intersections and only retains out-edges with remarkable heading changes. The rationale behind is that the number of such out-edges only take a quite small fraction. In addition, HCC are spatial-lossless, while MMTC, PRESS and CCF are all spatial-loss.

## 2.3 Basic Concepts and System Overview

### 2.3.1 Basic Concepts

**Definition 2.1** (Compressed Trajectory Segment). Given a mapped trajectory segment $\tau_m = \langle e_i, \cdots, e_n \rangle$, its compressed form is defined as $\{t_i, \tau_c\}$. $t_i$ is the starting time of this mapped segment. $\tau_c$ is a subset of $\tau_m$, denoted as $\tau_c = \langle e_i, \cdots e_m \rangle$, where $m \ll n$ and each pair of two consecutive edges is usually not connected in the road network. The compressed trajectory steam $T_c$ is composed of an unbounded set of compressed trajectory segments $\tau_c$ and their starting times, denoted by $T_c = \langle \{t_1, \tau_{c1}\}, \{t_2, \tau_{c2}\}, \cdots \rangle$.

**Definition 2.2** (Heading Change at Intersections). Heading change at intersections[2] is defined as the angle difference between the heading direction of the last GPS point (e.g., $p_1$) before the vehicle enters the intersection and the heading direction of the first GPS point (e.g., $p_2$) after leaving the intersection. According to the angle value, the heading change (0–360°) is divided into four categories, denoted as N (0–45°, 316–360°), E (46–135°), S (136–225°) and W (226–315°), as shown in Fig. 2.1. Heading changes belonging to E indicate that the vehicles ***go straight*** when passing intersections; heading changes belonging to other three categories mean that the vehicles ***make turns*** (e.g., left, right, or U) when passing intersections.

**Definition 2.3** (In-Edge and Out-Edge). In-edge and out-edge refer to the edge that the vehicle enters at (e.g., $e_1$) and leaves from (e.g., $e_2$) an intersection respectively, as illustrated in Fig. 2.1. The out-edge is determined by the in-edge and the heading change of the vehicle at the intersection.

To ease the computation of heading changes, we can simply use the angle difference between the in-edge and out-edge directions along the vehicle moving direction to approximate the heading changes at the given intersection. In this way, heading changes of mapped trajectory segments can be also estimated, since mapped trajectory segments don't save heading direction information any more.

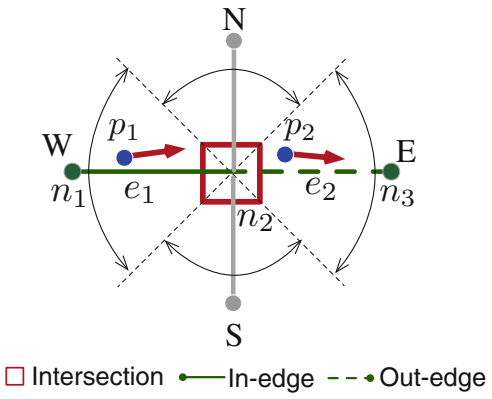


**Fig. 2.1** Illustration of heading change, in- and out-edges

### 2.3.2 System Overview

The framework of VTracer consists of three major units, namely vehicle-mounted GPS devices, mobile phones and data center, as shown in Fig. 2.2. The vehicle-mounted GPS devices have two main functions. For one thing, the devices collect the vehicular locations (GPS points) at a constant rate, then transmit raw trajectory stream (an unbounded set of GPS points) into mobile phones via Bluetooth. For another, the devices receive the compressed data from mobile phones via Bluetooth, then send them to the data center via GPRS. Mobile phones comprise two components, i.e., Map Matcher and Compressor.

Map Matcher takes raw GPS trajectory stream as input and divides it into an unbounded set of raw trajectory segments whose length is controlled by user-specified parameter (i.e., l). Moreover, two consecutive segments share a common point, i.e., the first GPS point of the latter segment is also the last GPS point of the former segment. Afterwards, Map Matcher maps each raw trajectory segment onto a sequence of connected edges (i.e., the mapped trajectory segment) in the road network (i.e., map-matching). Then, for each mapped trajectory segment, Compressor compresses it and return the compressed representation to GPS devices. The data transmission between mobile phones and GPS devices is via Bluetooth, which is free of charge and has smaller latency. GPS devices receive the compressed trajectory segments and upload them to the data center via GPRS, which costs the third-party bandwidth. At last, the data center may decompress and manage the data for further applications, such as the common when-and-where query service.

Our developed system differs from the prior research mainly in the following aspects.

1. We collect the trajectory data densely to ease map matching and increase the accuracy. On the other hand, we also implement a more map matching algorithm which fully utilizes the under-explored heading direction information [14].
2. We offload the resource-hungry computing tasks, including online trajectory mapping and compression to the nearby smartphones of drivers without incurring extra communication cost, bringing the smart idea of mobile edge computing.

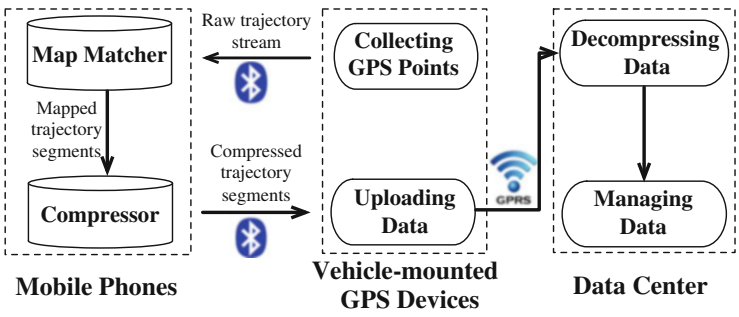


**Fig. 2.2** System overview of VTracer

3. The system can not only realize significant storage and communication savings, but also maintain a high-quality trajectory dataset in the data center for further applications.

## 2.4 HCC Algorithm

### *2.4.1 Motivation*

We first use the following motivating example to illustrate the basic idea of our proposed trajectory compression algorithm.

*Motivating Example: To recommend a unique route to drivers from the origin to the destination, instead of giving edge-by-edge guidance, GPS navigation systems usually only remind the drivers before changing driving directions at intersections (i.e., go straight, make left/right/U turn).*

Driving is a prudent thing. Thus, navigation systems actually provide a more concise trajectory representation, i.e., using heading changes at intersections to avoid distracting drivers when recommending driving directions. Inspired by the idea [1], present a Clockwise Compression Framework (CCF), which represents trajectory as the sequence of out-edge at each by-passing intersection from the origin to the destination. More specially, CCF encodes the out-edge with a unique code based on the ID of the intersection and the heading change. Excluding the starting and ending edges, the number of elements in the representation is just the number of by-passing intersections, which is smaller than the number of edges, occupying less space. There is a clear daily experience that we prefer to "going straight" more often at intersections during driving, meaning that straight out-edges are repeated in the trajectory. In such a situation, we can discard out-edges with small heading change to further reduce trajectory size. Thus, on top of the CCF, *we argue that the representation can be further reduced if retaining out-edges with remarkable heading changes ("making turns") only*. In such manner, compared to CCF, it is expected the number of elements in the compressed trajectory can be even smaller, which motivates us to propose the trajectory compression algorithm based on the heading change. To show the potential improvement that our proposed idea may achieve, we further provide statistics on the heading changes, including the percentages of "going straight" and "making turns" respectively. Based on our data, the percentages of heading changes belonging to "going straight" is over 92%, while the percentages of "making turns" is less than 8%. Therefore, the number of elements in the trajectory if represented by the out-edges with remarkable heading changes can be greatly reduced, expecting an improved compression performance. Note that the total number of changes in the statistical study is over 100,000 in the target city.

### 2.4.2 Algorithm Details

As discussed, rather than retaining all out-edges in a mapped trajectory segment, the key idea of our proposed HCC is to retain the out-edges when the vehicle makes a turn (or U-turn). Algorithm 2.1 summarizes the whole procedure of the proposed HCC algorithm. Note that the mapped trajectory segment ($\tau_m$) is the basic processing unit of HCC.

**Algorithm 2.1.** ***HCC*($\tau_m$, $G(N, E)$)**

```
1:  τ_c = e_1;
2:  for i = = 2 to | τ_m | − 1 do
3:     s = identifyNode(e_i, e_{i+1});
4:     if isIntersection(s, G(N, E)) then
5:        if ~isGoStraight(e_i, e_{i+1}) then
6:           τ_c = τ_c ∪ e_{i+1};
7:        end if
8:     end if
9:  end for
10: τ_c = τ_c ∪ e_{|τ_m|};
```

Line 1 in Algorithm 2.1 refers to the initialization of the compressed trajectory. More specially, HCC enrolls the first edge $e_1$ of the input trajectory into the compressed trajectory $\tau_c$. Lines 2–6 refer to the loop operation. In the loop, HCC algorithm scans and checks the remaining edges of the input trajectory one by one before it reaches the last edge. In more detail, for each edge $e_i$ in the input trajectory, HCC first identifies the node between $e_i$ and its following edge $e_{i+1}$ (Line 3), then judges whether it is an intersection node (Line 4). If yes, then it continues to identify the heading change of the vehicle at this intersection (Line 5). If the vehicle at the intersection is identified as NOT "going straight", we append the out-edge (i.e., $e_{i+1}$) in the previous obtained compressed trajectory $\tau_c$ (Line 6); otherwise, HCC just skips this edge and continues to process next one $e_{i+1}$. Finally, HCC enrolls the last edge $e_{|\tau_m|}$ into the compressed trajectory $\tau_c$ and terminates the whole procedure (Line 7). As can be observed, for any input mapped trajectory segment, HCC always retains the first and last edges in the compressed trajectory. The time complexity of HCC is $O(|\tau_m|)$, where $|\tau_m|$ is the number of edges in the input mapped trajectory $\tau_m$.

However, HCC may result in trajectory ambiguity, because of the complexity of the road network and the coarse granularity of the defined heading change category. For example, different out-edges may be identified with a same heading change. We use a simple example to illustrate the issue of trajectory ambiguity, as shown in Fig. 2.3. There are two mapped trajectories with the same starting and ending edges, e.g., $\tau_{m1} = \langle e_1, e_2, e_3, e_4, e_8 \rangle$ and $\tau_{m2} = \langle e_1, e_5, e_6, e_7, e_8 \rangle$. According to our proposed HCC algorithm, the compressed trajectory for both trajectories are the same, i.e., $\langle e_1, e_8 \rangle$, since the heading change between $e_1$ and $e_2$, and the heading change between $e_1$ and $e_5$ are both identified as "going straight". In this case, both out-edges would be discarded, which is incorrect and should be avoided.

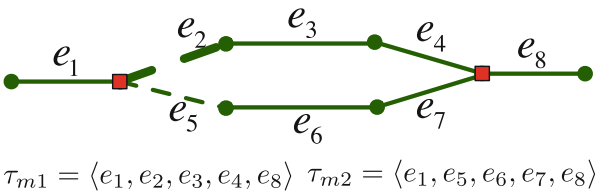


**Fig. 2.3** Illustrative example of trajectory ambiguity

To address the issue, there is an intuitive idea that HCC still maintains the out-edge in the compressed trajectory even it is identified as "going straight", if the case that more than one out-edge at the intersection are identified as "going straight" category occurs (Case I for short in the rest of presentation). For instance, with the intuitive idea, the compressed trajectories for the two trajectories shown in Fig. 2.3 are $\tau_{c1} = \langle e_1, e_2, e_8 \rangle$ and $\tau_{c2} = \langle e_1, e_5, e_8 \rangle$, respectively. However, such a method increases the number of edges that have to be maintained. To resolve trajectory ambiguity without costing much storage space, we embed the idea of Frequent Edge Compression (FEC) into HCC algorithm, detailed as follows.

Frequent Edge Compression: Similar to the observation that drivers are inclined to choose the shortest path from the origin to the destination, drivers may prefer to select some out-edge than others when traversing the intersections. Inspired by the daily experience, trajectories containing out-edges travelled by drivers frequently can be further reduced. The frequent out-edges are no need to be maintained. With the idea of FEC, HCC is expected to achieve even better compression performance. Taking the two trajectories shown in Fig. 2.3 as the example again. Suppose that $e_2$ out-edge is more popular than $e_5$, thus we can discard $e_2$ for the trajectory $\tau_{m1}$ during compression. Bringing in the idea of FEC, the compressed trajectories for $\tau_{m1}$ and $\tau_{m2}$ are $\tau_{c1} = \langle e_1, e_8 \rangle$ and $\tau_{c2} = \langle e_1, e_5, e_8 \rangle$, respectively. The improvement is significant since Case I happens quite commonly in our trajectory data.

### *2.4.3 Trajectory Decompression*

For a compressed trajectory, it is trivial to recover its original trajectory (i.e., the sequence of connected edges). Taking the example shown in Fig. 2.3 again, for the compressed trajectory $\tau_{c2} = \langle e_1, e_5, e_8 \rangle$, the first ($e_1$) and the last edge ($e_8$) corresponds to the starting and ending edge of the original trajectory respectively. The only remaining edge $e_5$ is the retained out-edge at the first intersection. There is no out-edge retained in the second intersection, which indicates that the vehicle generally goes through from $e_5$ to $e_8$. Hence, we can decompress the trajectory correctly. As a comparison, for the compressed trajectory $\tau_{c1} = \langle e_1, e_8 \rangle$, only the first and last edges are retained, which indicates that vehicles take the most common turn when travelling intersections in-between.

Although few semantics are embedded directly in the compressed trajectory, the trajectory decompression is capable of unveiling some common semantics by

coupling with other multi-source urban datasets [15]. To name a few, the average speed of the trajectory segment can be computed by dividing the total driving distance (the sum of each edge) to the total time. The average speed is a clear indicator of the traffic state [16]. A staying state can be safely claimed if the starting and ending edge are unique. Moreover, with the point-of-interest and digital map data, the surrounding spatial context of the trajectory (e.g., street names, the number of lanes) can be inferred [15].

## 2.5 Evaluations of HCC

In this section, based on the GPS trajectory data collected from taxis in the real world, we conduct extensive experiments to evaluate HCC algorithm in terms of compression ratio and computation time under different lengths of trajectory segment. We also test the performance of HCC algorithm on the quality of response for the when-and-where query.

### *2.5.1 Baselines*

We select three representative baselines to compare, i.e., PRESS [3], CCF [1] and DP algorithms [8], with their details as follows.

- With PRESS, the edge sequence between every pair of two consecutive edges in the compressed trajectory follows the shortest path exactly. Readers can refer to [3] for the details.
- From the starting edge to the ending edge, CCF only retains the out-edge ID and code (in the clock-wise order) at each intersection that the vehicle passes.
- DP aims at reducing the number of GPS points based on line-fitting. If the distance from the GPS point to the line segment is less than the error bound, then it will be discarded.

**Remark** PRESS and CCF are both based on the map matching and they are spatial-lossless, while DP is based on the raw trajectory segment (i.e., the sequence of GPS points) and spatial-lossy. Similar to HCC, for PRESS and CCF, the starting edge and ending edge of the trajectory segment are also always maintained. To accelerate the process of PRESS, the shortest paths for any two edges in the road network are usually pre-computed. However, it costs too much storage space (e.g., over 100 GB for Beijing City), which is intolerable in the mobile environment. Therefore, to make it comparable to HCC algorithm, we revise the original PRESS (i.e., the revised version) by adopting Dijkstra algorithm to achieve the online shortest path computation instead.

### 2.5.2 Experimental Setup

1. Data preparation: In the experiments, we prepare two different kinds of datasets from the city of Beijing, China. The basic dataset is the road network, which can be freely downloaded and extracted from OpenStreetMap. Totally, it contains 141,735 nodes and 157,479 edges. Moreover, OpenStreetMap also provides some additional attributes of edges, including number of lanes, number of traffic lights, speed limits and so on [17]. The second is the GPS trajectory data, which are generated by 595 taxis in one week (from 15th to 21st September, 2015). The sampling rate for all taxis is identical and constant, with a value of around 6 s. In terms of storage space, it takes up over 1.01 GB. Prior to compression, we used an SD-Matching algorithm to transform the raw trajectory data into a sequence of edges on the road network.
2. Evaluation metrics: For trajectory compression algorithms, the popular and well-known metric of compression ratio ($cr$) is used to quantify the compression effectiveness, which is defined as the ratio of the occupied storage space of the raw trajectory segments to the occupied storage space of the compressed trajectory segments. It is easy to understand that $cr$ should be always bigger than 1. The compression performance is better if $cr$ is bigger. We use the computation time cost at both trajectory mapping phase and compressing phase as the time cost when evaluating the efficiency for trajectory compression algorithms. Similarly, the average time is adopted to measure the overall efficiency.

There is an important user-specified parameter in HCC algorithm, i.e., the length of the trajectory segment ($l$). Thus, we study its effectiveness in terms of compression ratio and its efficiency in terms of time cost under different $ls$ in the following. As a comparison, the results of the baseline algorithms are also shown.

### 2.5.3 Varying l

As can be seen from Fig. 2.4a, HCC, PRESS and CCF algorithms achieve much higher compression ratio than DP algorithm under all $l$s, demonstrating the superior performance of map-matching based trajectory compression algorithms. HCC algorithm achieves the compression ratio in-between among all three map-matching based algorithms, and PRESS obtains the best performance. The reason why HCC performance better than CCF is that: CCF retains the out-edge information at every intersection, while HCC only retains out-edges with significant heading changes, which usually take up a small fraction of all out-edges. One credible reason why PRESS performs the best may be due to that taxi drivers prefer to taking the shortest path in real cases when delivering passengers [18]. Under such circumstance, the edges between the origin and destination can be commonly discarded, resulting in the best compression ratio. One major drawback of PRESS algorithm is that the shortest path between every two edges in the whole road network should be

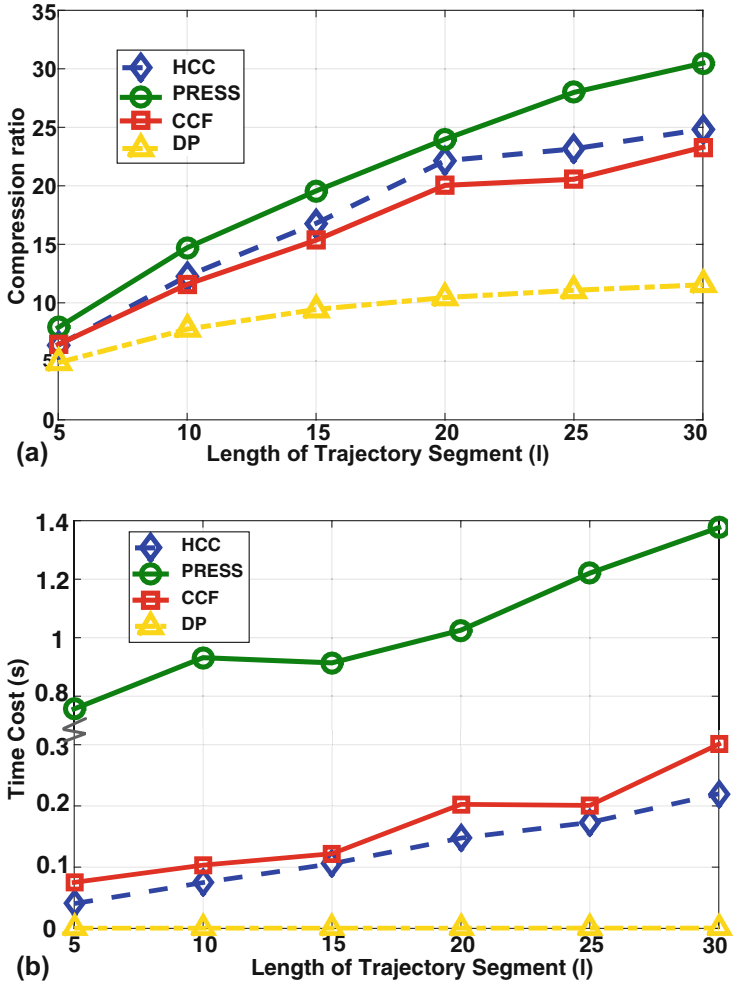


**Fig. 2.4** Comparison results of compression ratio (**a**) and time cost (**b**) for different algorithms under different *l*s

pre-computed and stored in order to save the compression time, which is always a time-consuming process. To make matters worse, the process must be repeated once the road network updates that is quite common in developing cities, causing many sustainable maintenance issues.

We also show the results of the time cost for HCC under different ls, as well as the results of the baseline algorithms in Fig. 2.4b. For all four algorithms, more computation time is required as the length of trajectory segment gets longer. DP algorithm is the most time-efficient because it needs no map-matching. For the other three map-matching based algorithms, as it can be predicted, PRESS needs much more computation time under all ls than the other three algorithms, since the online

shortest-path computation is time-consuming. HCC algorithm is more efficient than CCF algorithm. The reason is that, compared to HCC algorithm, CCF algorithm needs an additional operation of looking up out-edge code. In more detail, to ensure a timely response, for CCF algorithm, every out-edge at each intersection is encoded in the clockwise order and saved in a table in advance. The number of intersections in the road network is usually huge, thus the efficiency of the additional operation of table looking can be costly. It can be concluded that trajectory compression based on HCC algorithm make a nice trade-off between the compression ratio and computation time, when combining Fig. 2.4a, b.

### 2.5.4 *Boundary Choice of* l

We are also interested at the boundary choice of the length of trajectory segment (i.e., $l$), because the value of $l$ bigger, the performance of the proposed HCC algorithm better. However, more computation time is needed. In the boundary case, all trajectory segments should be compressed before the new sampling GPS is coming, that is, the maximum time cost of all trajectory segments should be less than the sampling time of the GPS trajectory data (i.e., 6 s in our study). Thus, to get the boundary choice of $l$, we increase $l$ with an equal interval (i.e., 10), and check the corresponding maximum time cost. We continue increasing the length until the maximum time cost reaches 6 s.

The maximum time costs under different $l$s are shown in Fig. 2.5a. The maximum time cost almost climbs linearly with the increase of $l$. The maximum time cost is close to 6 s when $l = 160$. We also examine the performance on the compression ratio by varying $l$ from 10 to 160. The improvement room is quite limited when $l$ increases from a large value. For instance, as shown in Fig. 2.5b, the compression ratio almost remains unchanged when $l$ increases from 120. Thus, it is no need to choose $l$ which consumes the time allowed (i.e., 6 s), and the boundary choice of $l$ is 120 in our case.

### 2.5.5 *Quality of Response for When-and-Where Query*

To support location-based services (LBS), some common types of query are generally applied on the top of the compressed trajectory data directly, among which when-and-where is the most popular query. For such query, users are mainly concerned with where the vehicle located at what time. To ensure the quality of response, the estimated and the actual locations of the vehicle should be as close as possible. The results of mean error under different lengths of trajectory segment ($l$) are shown in Table 2.1. From the table, we can observe that the mean error increase with the trajectory segment gets longer, due to the reason the total travel distance of the vehicle becomes longer with $l$ and the driving behavior is also more complex,

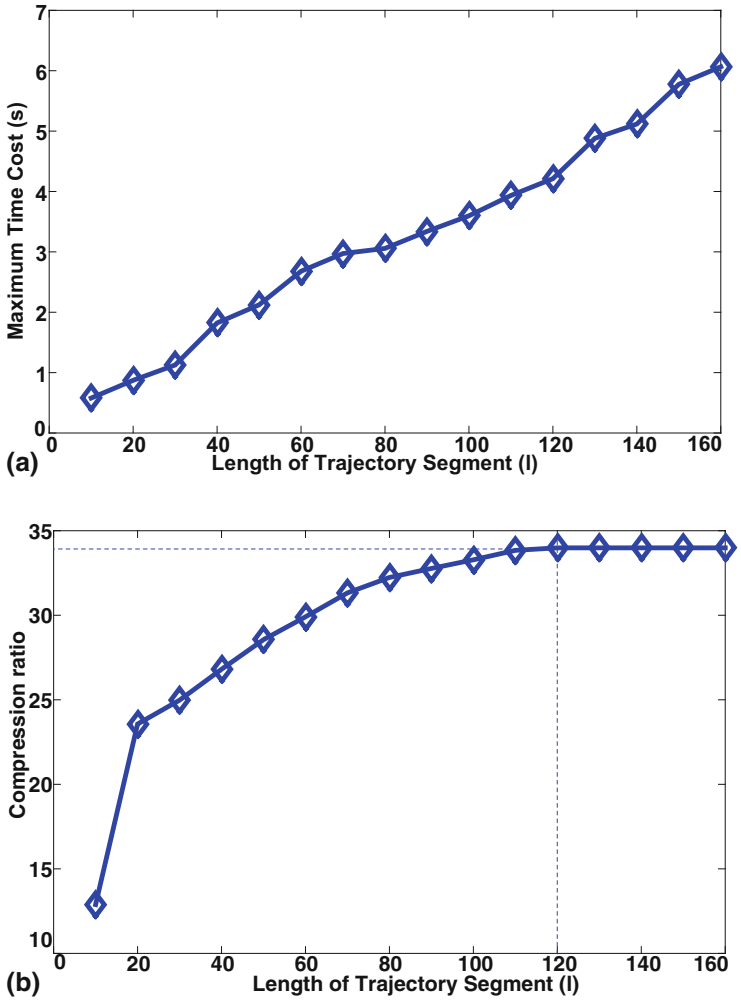


**Fig. 2.5** Results of the boundary choice of $l$. (**a**) Maximum time costs under different $l$s. (**b**) Compression ratios under different $l$s

**Table 2.1** Results of mean error in when-and-where query under different $l$s

| $l$s | 5 | 10 | 15 | 20 | 25 | 30 |
|---|---|---|---|---|---|---|
| Mean error (m) | 112.8 | 144.7 | 187.9 | 200.5 | 236.4 | 271.3 |

making it more difficult to estimate its position at the given time. However, compared to the results reported in [1, 3], our proposed algorithm still achieves a better query quality. For instance, the mean error of our method is less than 145 m when $l$ equals 10, which can be usable for most of LBSs in real life. We fix $l = 10$ in this experiment.

## 2.6 Evaluations of VTracer

In this section, we show that the compression results produced by VTracer are: first, with the high quality of trajectory mapping, second, with high compression ratio, and third, quite efficient and light-weight which can respond in the real time and work under the mobile environment.

### *2.6.1 Experimental Setup*

1. Deployment in a real environment: Figure 2.6 shows the deployment of VTracer system in the real environment. The system is deployed in a Volkswagen Passat Car made in 2015. There are two smartphones used in the deployed system. One of them is used to collect the GPS data, while the other is used to be the primary computing platform that is responsible for the heavy computation tasks. The models of smartphones used for data collection and data compression are Huawei P9 and P10, respectively. Huawei P10 has a RAM of 4 GB, which has strong computing capability. Bluetooth pairing between the two phones is required to establish the data communication before starting the system.
   We develop an app running on Android OS to control the built-in GPS sensor to mimic the vehicle-mounted GPS devices to collect the trajectory data. The sampling rate is set 6 s by default, which can be customized by programming. The data collector is mounted on top of the phone-holder to make sure that its heading direction is always consistent to the heading direction of the vehicle during driving, as highlighted in Fig. 2.6. The data stream is sent to the other

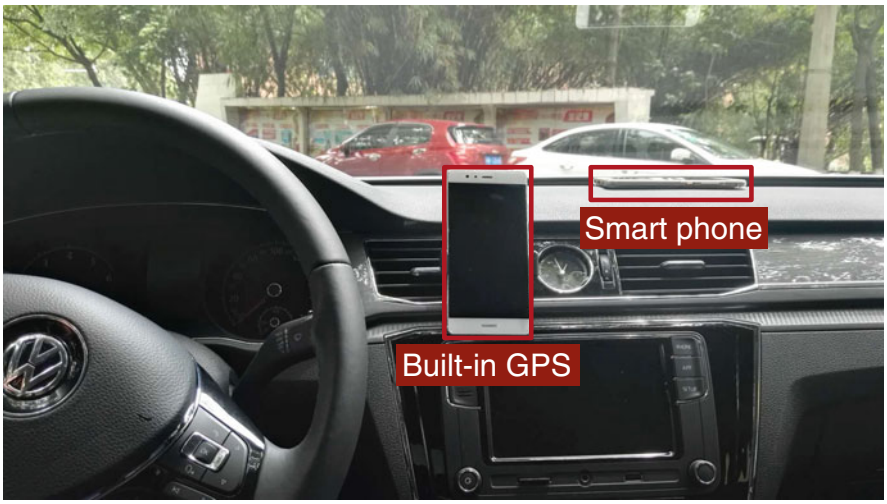


**Fig. 2.6** Deployment of VTracer system in real environment

smartphone via the established Bluetooth link. In the other smartphone, we also develop another app called *TrajCompressor* running on Android OS to compress the continuously received GPS data stream.

2. Data preparation: Two major datasets are prepared for the evaluation of VTracer in the city of Chongqing, China. The first one is the road network, which is downloaded from OpenStreetMap. Statistically, it totally contains 30,691 edges and 29,461 nodes. The second one is the raw GPS trajectory data containing around 7600 raw GPS points, which is collected from March 17th to April 21st, 2018. The accumulated driving distance of the trajectory dataset is around 513 km.
3. Baseline methods: We select two baselines to compare, i.e., PRESS and CCF. Moreover, we implement them in the same Android platform and develop the corresponding apps, namely PRESS and CCF.
4. Evaluation metrics: Two metrics are used to measure the effectiveness of VTracer. Specifically, the average matching accuracy (*acc*) used to quantify the quality of map-matching; and the compression ratio *(cr*) used to measure the performance of trajectory compression. We define *acc* as the difference between 1 and the ratio of the number of wrongly-matched GPS points to the total number of observed GPS points, as shown in Eq. (2.1).

$$acc = 1 - \frac{N_{wm}}{N_{total}} \tag{2.1}$$

where $N_{wm}$ and $N_{total}$ refer to the number of wrongly-matched GPS points and the total number of observed GPS points, respectively. A given GPS point is reported wrongly matched if it is not mapped to its true edge. It should be noted that we recruit three volunteers to manually label the "true edge" that each GPS point should locate by voting. Hence, it is challenging or even impossible to compute the value of map-matching accuracy "on-the-go" because the ground truth can only be known *ex post*.

In previous studies, *cr* is usually defined as the ratio between the disk space that the trajectory data occupy before and after applying data compression algorithms. However, as to our VTracer system, a data point is sent to the local computing center for processing immediately once produced. At the side of the computing center, it also never stores data in the hard disk. To make the computation of *cr* feasible, we approximate it using the following formula, as shown in Eq. (2.2).

$$cr = \frac{N_{total} \times c_1}{\lceil N_{total}/l \rceil \times c_2 + N_{edge} \times c_3} \tag{2.2}$$

where $N_{total}$ refers to the number of total GPS points that the local computing center received and processed; $\lceil N_{total}/l \rceil$ gets the rounding up value of $N_{total}/l$, and it refers to the number of trajectory batches; $N_{edge}$ is the number of retained edges in the compressed trajectory; $c_1$ is the value of constant bytes that a GPS point occupies,

which is roughly estimated; similarly, $c_2$ and $c_3$ refer to the constant values of bytes that the time and an edge occupy, respectively. In our experiment, we set $c_1 = 40$ and $c_2 = c_3 = 9$. The compression performance is better if $cr$ is bigger. Compared to $acc$, it is feasible to compute $cr$ "on-the-go" since we can monitor the values of $N_{total}$ and $N_{edge}$ in real time. To be more specific, we simply set two counters to obtain the number of received GPS points and the number of retained edges during trajectory compression, which enables us to show and refresh the value of $cr$ timely.

To measure how efficient that VTracer is, we count the total time cost at both phases including trajectory mapping and trajectory compression for each trajectory batch. Moreover, we use the average time value among all time costs for trajectory batches to quantify the overall efficiency of VTracer. In addition, the energy and memory that VTracer consumes are also used to reflect its efficiency.

### 2.6.2 Effectiveness Study

We are quite interested in the question, that is, *does VTracer perform consistently for all rides?* To address such issue, we intend to investigate the system performance for a sample of rides, in terms of the matching accuracy ($acc$) and the compression ratio ($cr$), respectively. Specifically, we first category all rides into five groups according to their driving distance, i.e., 0–10, 10–15, 15–20, 20–25, >25. Then, for all rides belonging to the same group, with VTracer, we obtain the average, minimum, and maximum values of the two performance indicators, as shown in Fig. 2.7. From results shown in two figures, we can draw the conclusion that the average performance is rather stable for rides with different driving distances. For instance, first, the accuracy of trajectory mapping is high and always above 95% for all rides; second, the compression ratio is also stable and above 15 for all rides. In addition, we can observe that, when the driving distance becomes longer, the range of both performance indicators become narrower, i.e., the system performs more stably. One possible explanation for such observation could be that, the vehicle may still stay within the same region (inner-region) with high confidence if driving shortly. Under this circumstance, the spatial context of the ride along the path may be very similar. Specifically, the road network along is either dense or sparse, which leads the performance to the extreme, i.e., either very poor or excellent. In contrast, the vehicle may cross different regions (inter-region) while driving far. In this case, the spatial context of the ride along the path may be averaged. Hence, to sum up, the performance can vary much more significantly for rides with smaller driving distance.

We also compare the compression ratio of VTracer to two baselines. The experiment results are shown in Table 2.2. As can be observed, VTracer achieves the compression ratio in-between among three systems, and PRESS obtains the best performance. The reason why VTracer performs better than CCF is that: CCF retains the out-edge information at every intersection, while VTracer only retains out-edges with significant heading changes, which usually take up a small fraction of all

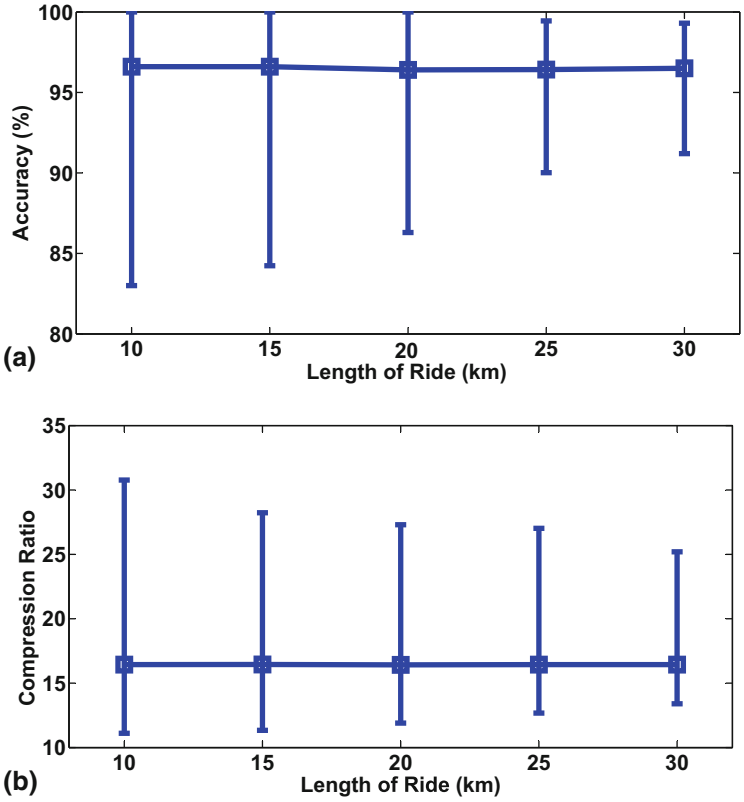


**Fig. 2.7** Results of *acc* and *cr* for rides belonging to different groups. (**a**) Accuracy of trajectory mapping. (**b**) Compression ratio of trajectory compression

**Table 2.2** Compression ratios of different trajectory systems

| System | VTracer | PRESS | CCF |
|---|---|---|---|
| cr | 15.85 | 16.67 | 15.69 |

out-edges. One credible reason why PRESS performs the best may be due to that drivers prefer to taking the shortest path in real cases [19]. Under such circumstance, the edges between the origin and destination can be commonly discarded, leading to the best compression ratio.

### 2.6.3 *Efficiency Study*

#### 2.6.3.1 Time Cost

Table 2.3 shows the results of the average and maximum time cost when three trajectory compression systems compress trajectory batches. As can be seen, VTracer consumes less computing time than CCF. The reason is that, compared to VTracer, CCF needs an additional operation of looking up out-edge code from the predefined database consisting of thousands of items. PRESS needs significantly more computation time compared to the other two systems, since PRESS needs multiple online shortest-path computations that are significantly time consuming. More specifically, the maximum time cost of PRESS is about 9 s, which is even larger than the sampling time (i.e., 6 s). We can safely draw a conclusion that VTracer is preferable for online trajectory compression system, because it strikes a nice tradeoff between the compression ratio and computation time cost when combining Tables 2.2 and 2.3.

#### 2.6.3.2 Energy Consumption

We adopt the electricity consumption (in the unit of mAh) to represent the energy consumption of mobile phones. In order to measure the energy consumption accurately, we just turn OFF all other unnecessary apps, then use an embedded app in Huawei Android OS to monitor the energy consumption of TrajCompressor app an hour. Figure 2.8 plots two curves of energy consumption of TrajCompressor app on the mobile phone with a working time duration of 12 h. One curve corresponds to the case when the visualizer is enabled while the other one corresponds to the case when the visualizer is disabled. It is easy to understand that more energy will be killed if enabling the front-end visualizer. Thus, the visualizer is disabled default to save energy. For both cases, from the figure, we can see that the energy consumption climbs almost linearly as the working time gets longer. As expected, the slop is much bigger when disabling the visualizer. The battery capacity of a HUAWEI P10 smartphone is 3000 mAh, so TrajCompressor can continuously work for around 140 h. In addition, it can work as long as about 280 h if the visualizer is disabled. In summary, the energy consumption of *TrajCompressor* app is acceptable in real-application scenarios.

As a comparison, we also show the energy consumption of the other two apps in Table 2.4. As can be predicted, TrajCompressor app is the most energy-efficient while PRESS app consumes the largest amount of energy, since a more computation time usually implies a higher consumption of electrical power.

**Table 2.3** Time cost of different trajectory compression systems

| System | VTracer | PRESS | CCF |
|---|---|---|---|
| Average time cost (ms) | 146 | 1483 | 359 |
| Maximum time cost (ms) | 400 | 9348 | 671 |

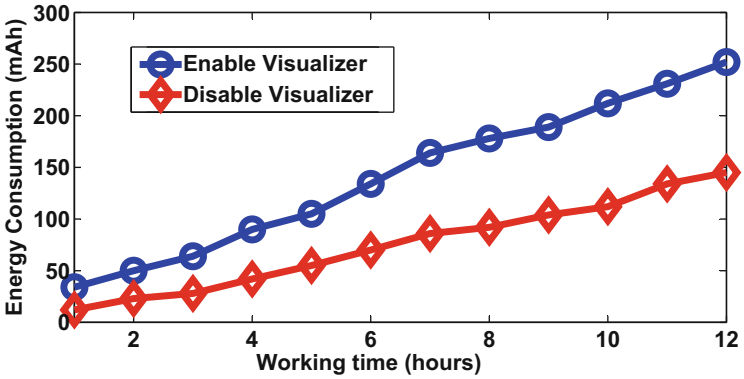


**Fig. 2.8** Energy consumption results of TrajCompressor app with respect to the working time when the visualizer is disabled and enabled

**Table 2.4** Energy consumption of different trajectory compression apps

| App | TrajCompressor | PRESS | CCF |
|---|---|---|---|
| Energy consumption (mAh) | 210 | 920 | 245 |

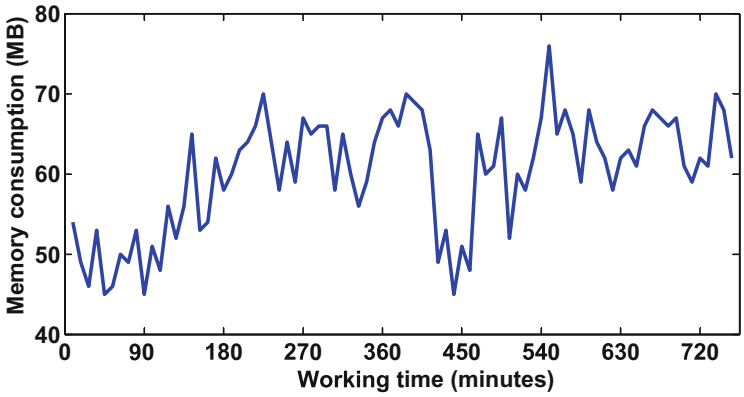


**Fig. 2.9** Memory consumption of *TrajCompressor* app with respect to the working time

### 2.6.3.3 Memory Consumption

Similar to the energy consumption study, we also rely on another embedded app in Huawei Android OS to monitor the memory consumption of *TrajCompressor* (in the unit of MB). Compared to the energy consumption, the memory consumption is more sensitive to the working time, we thus monitor this parameter every 10 min. Figure 2.9 shows the result of memory consumption of the mobile phone with a working time duration of 12 h. As can be observed, the memory consumption fluctuates with the working time. The maximum memory consumption is about

**Table 2.5** Memory consumption of different trajectory compression apps

| App | TrajCompressor | PRESS | CCF |
|---|---|---|---|
| Average memory consumption (MB) | 60 | 97 | 86 |
| Maximum memory consumption (MB) | 76 | 161 | 121 |

76 MB, which only takes up 2.0% of the whole memory (i.e., 4 GB). To sum up, the memory consumption is also acceptable and TrajCompressor almost does not impact other mobile apps.

We also record the memory consumption of the other two baseline apps, with the results (i.e., average and maximum memory consumption) shown in Table 2.5. Quite similar to the study of energy consumption, TrajCompressor app occupies less memory than the other two apps PRESS and CCF, because of the additional operation of looking up out-edge code (CCF) and computation of shortest-path (PRESS) will consume more computing resource of the mobile phone.

## 2.7 Conclusions and Future Work

In this chapter, we present a novel trajectory compression algorithm called HCC, and a novel system called VTracer, with the objective of compressing trajectory data before sending it to the data center. Compared to previous systems that collect trajectory data less frequently that are sensitive to GPS noises, we achieve such goal by collecting dense vehicle trajectory data at the side of GPS devices and sending the compressed trajectory data that is expected with less size but more completed and informative to the data center. Inspired by the idea of mobile edge computing, we migrate the heavy online trajectory compression task to the mobile phones of drivers. Extensive results in real deployment demonstrate the superior performances of our developed VTracer system in terms of the quality of map-matching, compression ratio, memory, energy consumption, and app size.

In the future, we plan to broaden and deepen this work in the following directions. First, we plan to take more measures to further improve compression performance, including compression enhancements and more advanced edge coding methods. Second, we also intend to investigate the trajectory compression in the temporal dimension. For instance, the accelerating (decelerating) behaviors of drivers may be retained, which reflect drivers' driving styles and traffic congestion at that time. To be more specific, we can easily calculate the mean accelerated speed ($a$) of adjacent two consecutive GPS points. Then a series of two-dimensional data ($t_i, a_i$) can be obtained, where $a_i$ refers to the accelerated speed of adjacent two points $p_i$ and $p_{i+1}$, and $t_i$ is the starting time of $a_i$. We could utilize some common methods (e.g. Huffman coding) to compress the two-dimensional data. It should be noted that trajectory data may be collected with equal time interval. Third, to enable more and smarter urban services, we plan to explore the potentials of the combination research of trajectory compression and trajectory summarization [20], due to the

capacity making the trajectory data more intuitively and more understandable of trajectory summarization. Last but not least, according to [21–23], we know that deep learning can translate trajectory into embedding representation with lower dimension, which is conductive to further services, e.g. traffic prediction, map matching. The result also inspires us to take it into consideration that combing trajectory compression with deep learning. For instance, we can design a trajectory embedding network, which is capable of representing trajectory compactly and recovering trajectory without spatial loss.

## References


1. Ji Y, Zang Y, Luo W, Zhou X, Ding Y, Ni LM. Clockwise compression for trajectory data under road network constraints. In: 2016 IEEE international conference on big data (big data), 2016. p. 472–81.
2. Muckell J, Olsen PW, Hwang JH, Lawson CT, Ravi SS. Compression of trajectory data: a comprehensive evaluation and new approach. GeoInformatica. 2014;18(3):435–60.
3. Song R, Sun W, Zheng B, Zheng Y. PRESS: a novel framework of trajectory compression in road networks. In: Proceedings of the Vldb endowment, 2014. p. 661–72.
4. Chen Y, Jiang K, Zheng Y, Li C, Yu N. Trajectory simplification method for location-based social networking services. In: Proceedings of the 2009 international workshop on location based social networks, 2009. p. 33–40.
5. Lou Y, Zhang C, Zheng Y, Xie X, Wang W, Huang Y. Map-matching for low-sampling-rate GPS trajectories. In: Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems, 2009. p. 352–61.
6. Sun P, Xia S, Yuan G, Li D. An overview of moving object trajectory compression algorithms. Math Probl Eng. 2016;2016:6587309, 1–13.
7. Shi W, Cao J, Zhang Q, Li Y, Xu L. Edge computing: vision and challenges. IEEE Internet Things J. 2016;3(5):637–46.
8. Douglas DH, Peucker TK. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica. 1973;10(2):112–22.
9. Han Y, Sun W, Zheng B. COMPRESS: a comprehensive framework of trajectory compression in road networks. ACM Trans Database Syst. 2017;42(2):1–49.
10. Ji Y, Liu H, Liu X, Ding Y, Luo W. A comparison of road-network-constrained trajectory compression methods. In: 2016 IEEE 22nd international conference on parallel and distributed systems (ICPADS), 2016, p. 256–63.
11. Silva A, Raghavendra R, Srivatsa M, Singh AK. Prediction-based online trajectory compression, 2016. p. 1–13.
12. Feng Z, Zhu Y. A survey on trajectory data mining: techniques and applications. IEEE Access. 2016;4:2056–67.
13. Kellaris G, Pelekis N, Theodoridis Y. Map-matched trajectory compression. J Syst Softw. 2013;86(6):1566–79.
14. Chen C, Ding Y, Xie X, Zhang S. A three-stage online map-matching algorithm by fully using vehicle heading direction. J Ambient Intell Humaniz Comput. 2018;9(5):1623–33.
15. Richter KF, Schmid F, Laube P. Semantic trajectory compression: representing urban movement in a nutshell. J Spatial Inform Sci. 2012;4:3–30.
16. Castro PS, Zhang D, Li S. Urban traffic modelling and prediction using large scale taxi GPS traces. In: 2012 International conference on pervasive computing, 2012. p. 57–72.

17. Chen C, Chen X, Wang L, Ma X, Wang Z, Liu K, Guo B, Zhou Z. MA-SSR: a memetic algorithm for skyline scenic routes planning leveraging heterogeneous user-generated digital footprints. IEEE Trans Veh Technol. 2017;66(7):5723–36.
18. Li B, Zhang D, Sun L, Chen C, Li S, Qi G, Yang Q. Hunting or waiting? Discovering passenger-finding strategies from a large-scale real-world taxi dataset. In: 2011 IEEE international conference on pervasive computing and communications workshops (PERCOM workshops), 2011. p. 63–8.
19. Castro PS, Zhang D, Chen C, Li S, Pan G. From taxi GPS traces to social and community dynamics: a survey. ACM Comput Surv (CSUR). 2013;46(2):17, 1–34.
20. Andrae S, Winter S, Strobl S, Blaschke T, Griesebner G. Summarizing GPS trajectories by salient patterns, 2005.
21. Shen Z, Du W, Zhao X, Zou J. DMM: fast map matching for cellular data. In: Proceedings of the 26th annual international conference on mobile computing and networking, 2020. p. 1–14.
22. Zhao K, Feng J, Xu Z, Xia T, Chen L, Sun F, Guo D, Jin D, Li Y. DeepMM: deep learning based map matching with data augmentation. In: Proceedings of the 27th ACM SIGSPATIAL international conference on advances in geographic information systems, 2019. p. 452–5.
23. Cao H, Xu F, Sankaranarayanan J, Li Y, Samet H. Habit2vec: trajectory semantic embedding for living pattern recognition in population. IEEE Trans Mobile Comput. 2020;19(5):1096–108.