VTracer: when online vehicle trajectory compression meets mobile edge computing

Chao Chen, Yan Ding, Zhu Wang, Junfeng Zhao, Bin Guo, Daqing Zhang

VTracer: When Online Vehicle Trajectory Compression Meets Mobile Edge Computing

Chao Chen, Yan Ding, Zhu Wang, Junfeng Zhao, Bin Guo, and Daqing Zhang, *Fellow, IEEE*

***Abstract*—Vehicles can be easily tracked due to the proliferation of vehicle-mounted global positioning system (GPS) devices. $VTracer$ is a cost-effective mobile system for online trajectory compression and tracing vehicles, taking the streaming GPS data as inputs. Online trajectory compression, which seeks a *concise and (near) spatial-lossless* data representation before revealing the next vehicle's GPS position, is gradually becoming a promising way to alleviate burdens such as communication bandwidth, storing, and cloud computing. In general, an accurate online map-matcher is a prerequisite. This two-phase approach is nontrivial because we need to overcome the essential contradiction caused by the resource-constrained GPS devices and the heavy computation tasks. $VTracer$ meets the challenge by leveraging the idea of mobile edge computing. More specifically, we offload the heavy computation tasks to the *nearby* smartphones of drivers (i.e., smartphones play the role of cloudlets), which are almost idle during driving. More importantly, they have relatively more powerful computing capacity. We have implemented $VTracer$ on the Android platform and evaluate it based on a real driving trace dataset generated in the city of Chongqing, China. Experimental results demonstrate that $VTracer$ achieves the excellent performance in terms of matching accuracy, compression ratio, and it also costs the acceptable memory, energy, and app size.**

***Index Terms*—Global positioning system (GPS) devices, mobile edge computing, resource-constrained, trajectory mapping, trajectory compression.**

I. Introduction

THE wide proliferation of various global positioning system (GPS) devices and mobile Internet in daily life has made many kinds of trajectory data easily available at a large scale, among which the vehicle trajectory data is a typical representative [6], [29]. Such trajectory data can not only be used to track the traveling paths and positions of vehicles directly, but also can be mined to enable a plenty of smart pervasive and urban services, such as understanding urban/traffic dynamics [28], suggesting driving routes [8], [10], inferring urban/building functionality [5]. Thus, increasingly more attention has been paid to mining trajectory data for urban computing tasks during recent years. To save communication and storage costs, vehicles simply report their GPS locations to the data center at a sparse time interval, i.e., less frequently. For instance, taxi-mounted GPS devices send their locations to the data center at a time interval of 1 or 2 min according to whether the taxi is occupied by passengers or not [7]. Unfortunately, this simple and native solution also raises some severely negative issues, e.g., creating a big uncertainty between two consecutive GPS points [11]. To make matter worse, GPS noises and the complexity of road network make the inference of driving paths in-between even more challenging [6]. As a result, the applications that can be enabled are limited to some extent.

To strike a tradeoff between cost savings and the usability of trajectory data, a promising way is the online trajectory data compression, that is, *by collecting the GPS locations more frequently at the side of GPS devices but uploading less yet completed data to the data center.* Thus, the computation of *concise and spatial-lossless* trajectory representation is essential. However, such computation is usually resource-hungry and vehicle-mounted GPS devices themselves just cannot afford that burdensome task. In addition, to make the idea feasible, there are some requirements that the system should satisfy, detailed as follows.

1) Computing Capability: In general, online trajectory mapping is a prerequisite for high-performance trajectory compression, which is also computation-intensive. Therefore, it needs a powerful computing platform. Unfortunately, the vehicle-mounted GPS devices do not suit due to the limited random access memory (RAM). For instance, the embedded RAM is only 4 kB for current taxi-mounted GPS devices [17].

2) Low Latency: The position data of moving vehicles are collected *densely and continuously* at the side of GPS devices. Thus, to ensure that the system can process the online trajectory data compression timely, low latency is necessary, i.e., the trajectory data in the buffer should be compressed before the new data point comes. Moreover, the lower time interval of consecutive GPS points (the sampling time interval) calls for the lower latency of the system.

Manuscript received March 13, 2019; revised August 12, 2019; accepted August 12, 2019. This work was supported in part by the National Key Research and Development Program of China under Grant 2018YFB1004403, in part by the National Natural Science Foundation of China under Grants 61872050 and 61602067, in part by the Fundamental Research Funds for the Central Universities under Grants 2018cdqyjsj0024 and 2019cdxyjsj0022, and in part by the Chongqing Basic and Frontier Research Program under Grant cstc2018jcyjAX0551. *(Chao Chen and Yan Ding contributed equally to this work.) (Corresponding author: Chao Chen.)*

C. Chen and Y. Ding are with the Key Laboratory of Dependable Service Computing in Cyber Physical Society (Ministry of Education), Chongqing University, Chongqing 400044, China and also with the College of Computer Science, Chongqing University, Chongqing 400044, China (e-mail: ivanchao.chen@gmail.com; duke_ding@sina.cn).

Z. Wang and B. Guo are with the Department of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China (e-mail: transitwang@gmail.com; guobin.keio@gmail.com).

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

D. Zhang is with Institut Mines-TELECOM/TELECOM SudParis, 91000 Evry, France (e-mail: daqing.zhang@telecom-sudparis.eu).

Digital Object Identifier 10.1109/JSYST.2019.2935458

3) Light Weight: The system should be light-weight, since the vehicles are moving and the room for the computing embedded is also limited in the mobile environment. In addition, a light-weight system generally consumes less energy, the supply of which is also usually limited in the mobile environment.

A. Our Ideas

To meet the above-mentioned system requirements, we propose the 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* [21]. More specifically, First, Smartphones are well-suited for burdensome computing tasks, since they have relatively strong computing capacity and are almost idle while driving. Second, the dense trajectory data collected by the GPS devices can be easily offloaded to the smartphones wirelessly with negligible latency. For instance, it only costs less than 0.2 ms to send a data point from GPS devices to mobile phones via Bluetooth. Such time can be ignored compared to the sampling time intervals of GPS devices that is usually in the range of 1–10 s even when densely sampled. More importantly, we further propose a cost-effective online trajectory compression algorithm to respond timely. Third, we implement the online trajectory compression system as an app running on the Android smartphone platform. The app is so lightweight that it only occupies a little space and memory overhead. It is also energy-efficient and consumes less than 10% battery power for 12-h runtime. Besides, we further develop a *front-end visualizer*, which could well and intuitively inform drivers about their trajectories together with nearby spatial context (e.g., point of interest). The visualizer can be enabled or disabled according to the driver's preference.

Why using smartphones as the edge devices? An edge device is defined as any computing or networking resource residing between data sources (e.g., raw GPS trajectory) and cloud-based data centers [21]. In the application scenarios of intelligent transportation systems, common edge devices include smartphones, small-cell base stations (BSs), road-side units (RSUs) and Wi-Fi routers. For instance, Shi *et al.* [21] mention that a smartphone can play the role of an edge device, sitting between body sensors and the cloud. Similarly, Yi *et al.* [27] also migrate resource-hungry and energy-consuming computing tasks from wearable devices to mobile phones, since wearable devices are commonly constrained by computing capability and energy-supply. On top of the above-mentioned examples, we safely draw a conclusion that *it is common to offload some burdensome computing tasks from devices with weak computing capability to smartphones, which play the role of cloudlets.* As for other devices (e.g., RSUs and BSs), they have more powerful computing capability compared with mobile phones. As a result, the response time of the system could be reduced if adopted. However, they are not suitable as edge devices in our proposed system. Specifically, for RSUs, a key drawback is the intermittent connection due to the limited communication range and the sparse deployment of infrastructures [26]. Existing infrastructures cannot strongly support the online trajectory compression service, especially when massive vehicles simultaneously upload their trajectory data. For BSs, the high cost of communication between the vehicle-mounted GPS devices and the BSs will inevitably give rise to the total cost, which goes against our goal. Worse still, frequent handoff occurrences at BSs caused by the high mobility of vehicles may impede the trajectory compression progress [2], [3]. On the contrary, the use of smartphones can not only overcome the above-mentioned issues, but also avoid the scalability issue since the proposed system runs in the smartphone of each driver distributively.

B. Our Contributions

This article describes the design considerations, implementation details, and experimental evaluations of $VTracer$ system. 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), by taking advantages of smartphones fully. By using $VTracer$, the data center can obtain a noise-free, more concise yet still completed trajectory representation for the moving vehicles. Besides, the significant savings on storage, communication, etc., can also be achieved. Finally, we not only test the system performance of $VTracer$ but also compare it with the state-of-the-art techniques 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 of 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 $VTracer$.

II. Related Work

In this section, we briefly overview the related work on trajectory mapping, trajectory compression, and mobile edge computing. More importantly, we also highlight the differences of our $VTracer$ system from the prior research.

A. Trajectory Mapping

Quite a few of algorithms on trajectory mapping have been proposed [6], [12]. Previous work mainly focuses on finding the true positions of GPS observations, since the GPS sampling frequency is high and the gap between two GPS observation is quite narrow. As a result, the mapped trajectory can be easily identified once addressing the GPS measurement errors. *For dense trajectory data, the quality of trajectory mapping is generally high.* However, the dense trajectory data also brings other side effects, such as storage and communication costs. Hence, GPS devices simply report locations less frequently, resulting in a bigger distance gap between two sampling positions. In general, trajectory mapping for sparse trajectory data is more challenging. To get the high-quality trajectory mapping results, additional information including road network topology, road attributes, motion laws, and more advance inferring techniques such as probabilistic algorithms are proposed [4]. Unfortunately, the mapping quality is still not satisfactory when dealing with trajectory data with low sampling rate. For instance, the accuracy of the state-of-the-art mapping algorithms (e.g.,

ST-Matching [18] and SnapNet [19]) is less than 85% when the sampling time is bigger than 120 s.

B. Trajectory Compression

There is also quite a lot of work has been done to represent trajectory using less data [13], [15], [22], [30]. Trajectory compression algorithms can be roughly categorized into two groups, i.e., line-simplification-based and map-matching-based, according to the utilization of the road network [9], [16]. The latter group is more relevant to our work. The group of compression methods generally maps raw trajectory data onto the road network before compression. The utilization of road network endows the trajectory with more correct and semantical information, which further benefits the trajectory compression. The representative algorithms belonging to this group are MMTC [15], PRESS [22], CCF [14], and so on. In general, the trajectory compression algorithms are usually resource-hungry.

C. Mobile Edge Computing

To address issues including the bandwidth cost and the response time, the raw data are commonly transmitted and processed on edge devices at the *proximity* of data generators. For the transmission between edge devices and data generators, the common communication interface is generally classified into two kinds, i.e., the wireless network (e.g., Bluetooth and Wi-Fi) and cellular network (e.g., 3G, 4G, and 5G), according to the communication cost [1], [20], [23]–[25]. The wireless network is both free of charge and high speed, however, it is limited to a small communication range. For instance, the effective range of Bluetooth is commonly less than 10 m. On the other hand, although the use of cellular network will cause extra communication cost, some edge devices that are more capable of computing can be well connected, such as BSs and RSUs. These devices are usually used as edge devices to process burdensome computing tasks, such as analyzing road conditions, data offloading from high-speed vehicles. With such powerful edge devices, the response time could be greatly reduced. However, leveraging these devices may also bring in some other issues, such as the intermittent connection of RSUs and frequent handoff occurrences at BSs.

D. Differences

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

1) We collect the trajectory data *densely* to ease the trajectory mapping and increase the accuracy. On the other hand, we also implement a more advanced trajectory mapping algorithm that fully exploits the under-explored heading direction information [6].
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.

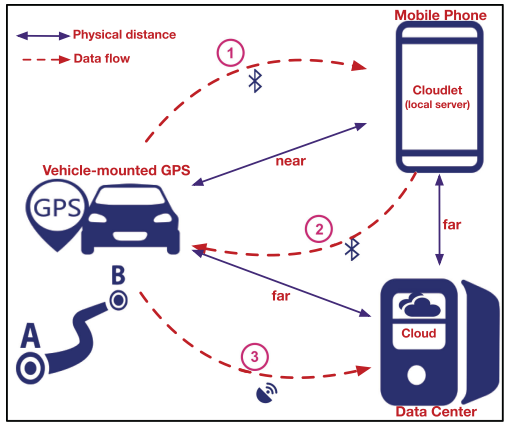


Fig. 1. System overview of $VTracer$.

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

III. System Overview

We now describe the design of $VTracer$. Fig. 1 shows the system overview. As can be observed, it consists of three components, i.e., *vehicle-mounted GPS devices, mobile phones, and the cloud-based data center*. The physical distance between GPS devices and the mobile phones is *stable and short*, while it is *remote and changeable* between GPS devices and the data center due to the high mobility of vehicles. GPS devices collect the trajectory data from moving vehicles, and send the data to the mobile phones of drivers via Bluetooth continuously (the data flow labeled as ① in the figure). Taking the raw trajectory data stream as inputs, the developed app called $TrajCompressor$ running on smartphones handles the online trajectory compression for each trajectory batch sequentially, and returns the compressed data batches back to the GPS devices, again via Bluetooth (the data flow labeled as ② in the figure). Note that the communication between GPS devices and mobile phones via Bluetooth is high-speed and free of change. Finally, the compressed trajectory data that are expected to be much smaller but still informative are uploaded to the data center (the data flow labeled as ③ in the figure) for the further usage via the third-party bandwidth (GPRS, 3G, 4G, etc.). As predicted, a significant communication saving can be also achieved. We will introduce the details of three components as follows.

A. Vehicle-Mounted GPS Device

The GPS device collects the locations of moving vehicles at a high sampling frequency, e.g., every 6 s. Initially, the GPS device sends such dense trajectory data to the mobile phones for data compression. Afterwards, the GPS device uploads the compressed trajectory data getting from the mobile phone to the data center. Compared to the traditional method, which simply collects vehicles' positions less frequently at the side of GPS devices without data compression, it is expected that $VTracer$ is more informative and robust to GPS noises.

B. Mobile Phone

As discussed, the mobile phone play as the role of local computing server since it is more capable of computing. Specifically, the mobile phone is mainly responsible for the two resource-hungry computing tasks, i.e., the online trajectory mapping and trajectory compression. Also, in order to make the drivers be informed about their traveling paths and surrounding spatial context (e.g., points of interests), we further develop a front-end visualizer to display the relevant information on the screen by recalling APIs provided by AMap.[1] Note that such visualizations can be simply disabled to save energy consumption.

C. Cloud-Based Data Center

The data center is the warehouse of storing the compressed trajectory data generated from city-scale vehicles in support of a wide range of innovative urban services, such as location-based services, vehicle tracking, and data visualization.

IV. Online Trajectory Mapping Algorithm

$VTracer$'s algorithm for map-matching differs from previous approaches in exploiting a new dimension of collected GPS data (i.e., the heading direction) systematically. The goal of the algorithm is to associate a sequence of GPS points to a sequence of connected edges on a known road network.[2] The data stream is usually split into nonoverlapped trajectory batches with equal size (the number of GPS points in each batch is same). In the implementation, we set batch size to 10 (i.e., $l = 10$). For each trajectory batch, to get its mapped trajectory, we propose a three-stage approach that utilizes the heading direction information completely, i.e., identifying top-k candidate edges for a given GPS point, path-finding for two consecutive GPS points in-between, and path-refining for the given trajectory batch. At the stage of finding the true position for each GPS point, the difference between the heading direction and the edge direction is used as an important indictor when identifying the correct located edge. At the stage of path-finding between two consecutive GPS points, the heading directions at the GPS points are used to narrow the search space over the road network. The heading direction difference between the two GPS points is used as a guider to find the true path efficiently. At the stage of refining path, each heading direction within the trajectory batch is counted globally and used to compute the likelihood of each possible mapped trajectory. Readers can refer to [6] for more technical details. Due to space limitation, here we only briefly introduce the skeleton of the algorithm, with relatively more emphasis on the implementation details.

A. Stage 1: Identifying Candidate Edges

At this stage, for a given GPS point, it usually does not seat on the road edge ideally due to the measurement error. To identify the top-k most likely edges that it locates, we take both the *distance* between the point to the "nearby" edges and the *angle difference* between the heading direction at the point and the nearby edge directions into consideration. The probability of a nearby edge (with a distance less than 100 m from the point) being the truly located edge is computed based on the defined *distance and angle difference* collectively. *The computation bottleneck at this stage is the determination of nearby edge set to the given GPS point.*

Intuitively, to determine such a set, we need to compute the distances from the point to all edges in the road network. However, it is extremely inefficient since the number of edges in a city is huge, which should be avoided. To accelerate the process, here we adopt a simple grid indexing mechanism. In more detail, the whole city is divided into 30×30 equal-sized grid cells. The area for each grid is around 1.69 km^2. For each cell, it is easy to know its located nodes and edges (i.e., road network fragment). The road network fragment is much smaller. Also, it is trivial to get the cell ID that the given point locates according to its longitude and latitude coordinates. Taking the road network fragment as inputs, it should be more efficient to determine the edge set, and the top-k candidate edges can be further identified as well. We set $k = 6$ in the implementation. In addition, *for all GPS points in the trajectory batch, their corresponding road network fragments are unified and saved in the memory of smartphones, for the further usage of trajectory compression.*

B. Stage 2: Path-Finding

The path that connects two consecutive GPS points would be found at this stage. For each pair of two connective GPS points, we need to find at most k^2 paths in total, since there are k^2 pairs of starting and ending edges. To improve the path-finding efficiency, we propose to take advantage of heading direction again to narrow the searching zone and guide the path-finding simultaneously. We implement this stage following the description in [6] rigidly.

C. Stage 3: Path-Refining

This stage aims at finding the mapped trajectory for the given trajectory batch. There would be $k^{2(l-1)}$ possible mapped trajectories. The final mapped trajectory is the one with the highest accumulated possibility [6]. However, the number is huge even if we select a small value of k and l. Therefore, it is impossible to enumerate all possible trajectories. Actually, as discussed in [6], due to the road network constraint, the actual number should be much smaller.

In the implementation of path-refining, we first construct a directed tree in which the node is the candidate edges for each GPS points. The directed edge in the tree is the path discovered in Stage 2 that connects two candidate edges. Then, on top of the built tree, we apply a simple tree pruning strategy to get a much tighter one by removing *invalid* nodes *iteratively*. A node is claimed *invalid* if its in-degree or out-degree equals 0, except for the starting and ending nodes.

[1] http://lbs.amap.com

[2] It can be freely downloaded from http://www.openstreetmap.org/

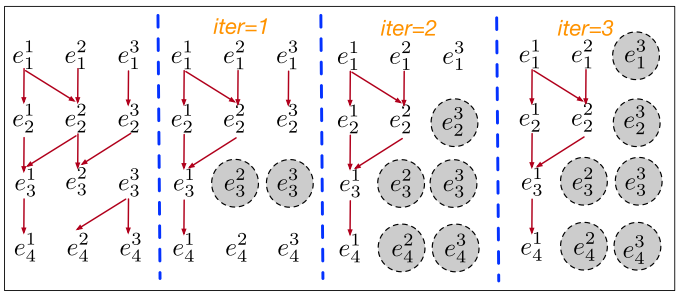


Fig. 2. Illustrative example on simple graph pruning strategy.

Algorithm 1: Tree Pruning Strategy.

1: $A(T)$ //Create an adjacent matrix for tree T
2: **while** $|A(T)|$ continues to reduce **do**
3: **if** $IsIndegreeZero(n_i)$ **then**
4: $T = T - n_i$; //Remove the node n_i from T
5: $Update(A(T))$;
6: **end if**
7: **if** $IsOutdegreeZero(n_j)$ **then**
8: $T = T - n_j$; //Remove the node n_j from T
9: $Update(A(T))$;
10: **end if**
11: **end while**

Algorithm 1 illustrates the procedure of tree pruning strategy. We first construct an adjacent matrix for the input tree (line 1). Lines 2–11 show the iterative operations of invalid node identification and removal. Iteration will be terminated when there is no newly invalid node can be found (i.e., $|A(G)|$ keeps unchanged, line 2). In the loop, after discovering the invalid node (lines 3 and 7), it will be removed from the tree and the corresponding adjacent matrix will be also updated (lines 4 and 5 and lines 8 and 9). It should be noted that when removing a node from the tree, its connecting edges are also removed simultaneously. Finally, all possible mapped trajectories are found by using *depth-first-search* algorithm on top of the reduced tree, and the one with the highest accumulated probability would be picked as the final result [6].

To make our idea clear, we use an illustrative example, as shown in Fig. 2. For better demonstration, we set l and k to 4 and 3, respectively. The depth of the tree is exactly equal to the size of the trajectory batch (i.e., l). The first level corresponds to the first GPS point while the last level corresponds to the last GPS point. The number of nodes in each level is identical, which just equals to the number of identified candidate edges (i.e., k). There does (not) exist an arrow if a path is (not) returned for a pair of candidate edges belonging to two consecutive GPS points (Stage 2). In the example, three iterations are needed in total. In the first iteration, nodes e_3^2 and e_3^3 will be removed as they are identified as invalid nodes. In the second iteration, three new nodes will be identified invalid and removed (e_2^3, e_4^2, and e_4^3). In the last iteration, e_1^3 will be removed. As can be seen, compared to the original tree, the newly reduced tree is much simpler, which should make the path-finding easier and faster.

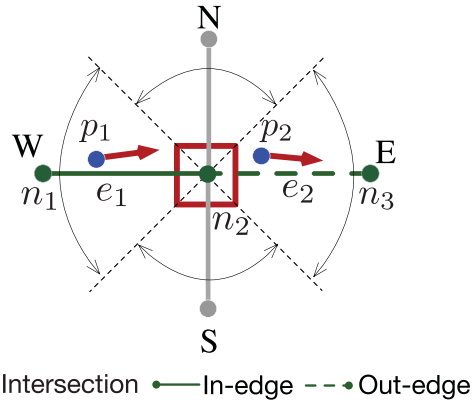


Fig. 3. Illustration of heading change, in- and out-edges.

V. Online Trajectory Compression Algorithm

The objective of the online trajectory compression is to remove some unnecessary edges in the mapped trajectory. In another word, we seek a concise trajectory representation using only some representative edges, which can be used to recover the original representation. Our algorithm takes following as input.

1) *A mapped trajectory batch and its starting time, denoted by* $T_m =< t_i, e_i, e_{i+1}, \ldots, e_n >$. Each pair of two consecutive edges in T_m is connected. Such representation returned by the trajectory mapping is redundant and prepared to be further compressed.
2) *A set of road network fragments.* It records the road network in a small geographic area where the vehicle may travel for each given trajectory batch, obtained at Stage 1 of the trajectory mapping algorithm. The scale of the obtained road network fragments is much smaller.

Compared to other popular algorithms, our proposed compression method does not need other auxiliary data such as the table of shortest paths in PRESS [22]. Its rationale is based on the edge angle itself. The output is a subset of T_m, denoted as $T_c =< t_i, e_i, \ldots, e_m >$, where m is much smaller than n in T_m. Note that T_c at least contains the first edge e_i of T_m. Each pair of two consecutive edges in T_c is usually not connected in the road network.

Before going to the algorithm detail, we define two key concepts, *heading change at intersections* and *in-edge and out-edge*, respectively.

Definition 1 (Heading change at intersections): It 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.[3] According to the angle value, the heading change ($0^\circ - 360^\circ$) is divided into four categories, denoted as N ($0^\circ - 45^\circ, 316^\circ - 360^\circ$), E ($46^\circ - 135^\circ$), S ($136^\circ - 225^\circ$), and W ($226^\circ - 315^\circ$), as shown in Fig. 3 . Heading changes belonging to E refer to that the vehicles *go straight* when passing intersections; heading changes belonging to the other three categories refer to that the vehicles *make turns* (e.g.,

[3]For brevity, we simply use "heading change" in the rest of presentation.

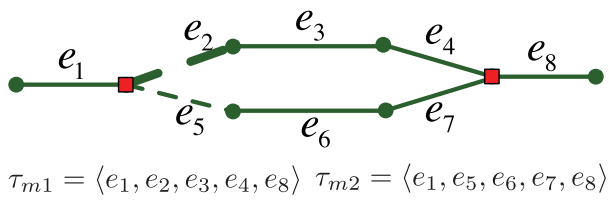


Fig. 4. Illustrative example of trajectory ambiguity.

left, right, or U) when passing intersections. Note that in the implementation, the degree of heading change takes an integer as a unit. Specifically, for a given degree, it will be rounded off to the nearest integer.

Definition 2 (In-edge and out-edge): They 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. 3. The out-edge is determined by the in-edge and the heading change of the vehicle at the intersection.

The algorithm contains three components, i.e., *identifying intersection*, *removing edges*, and *avoiding trajectory ambiguity*. To be more specific, the algorithm enrolls the first edge e_1 of the input trajectory into the compressed trajectory T_c. Thereafter, it 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, we first identify the node connecting e_i and its next edge e_{i+1}, then judge whether it is *an intersection node*, according to the saved road network fragments. Comparing to scan the road network in the whole city, the number of nodes in the subset is much less, leading to a significant searching time cost. If the node is identified as an intersection one, then we continue to compute the heading change of the vehicle at that intersection. If such heading change is NOT "going straight," we simply append the out-edge (i.e., e_{i+1}) in the previous obtained compressed trajectory T_c; otherwise, the algorithm just skips this out-edge and continues to process the next edge e_{i+1}. Finally, the algorithm enrolls the last edge $e_{|T_m|}$ into the compressed trajectory T_c and terminates the whole procedure.

However, only two components in the algorithm may cause *trajectory ambiguity*, due to the complexity of the road network and the coarse granularity of the defined heading change category. For instance, different out-edges may be identified with the same heading change. We use a simple example to illustrate the issue of trajectory ambiguity, as shown in Fig. 4. There are two mapped trajectories with the same starting and ending edges, e.g., $T_{m1} = \langle e_1, e_2, e_3, e_4, e_8 \rangle$ and $T_{m2} = \langle e_1, e_5, e_6, e_7, e_8 \rangle$. According to the first two components, 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. To address the issue, we choose an intuitive method is that we still maintain 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 is identified as "going straight" category occurs. For instance, with the idea, the compressed trajectories for the two trajectories shown in the Fig. 4 are $T_{c1} = \langle e_1, e_2, e_8 \rangle$ and $T_{c2} = \langle e_1, e_5, e_8 \rangle$, respectively.

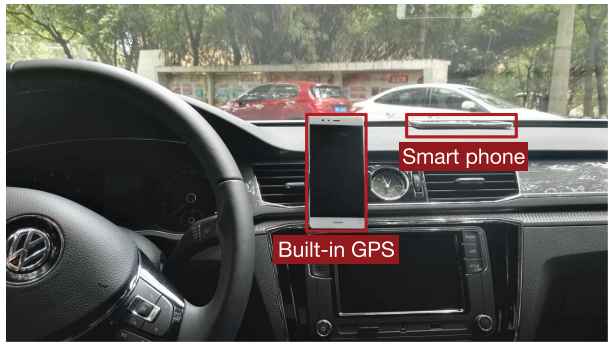


Fig. 5. Deployment of $VTracer$ system in the real environment.

VI. Evaluation

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. Moreover, we also conduct a case study to investigate the performance in terms of trajectory mapping and compression under different densities of the road network.

A. Experimental Setup

1) Deployment in a Real Environment: Fig. 5 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. 5. The data stream is sent to the other 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. The source code and the development manual are available at GitHub via the link.[4]

[4]https://github.com/MrBaixg/TrajCompressor-STM32-Android-GPS

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.[5] 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 compare our proposed algorithm to two baselines, i.e., PRESS and CCF. In addition, we implement them in the same Android platform and develop the corresponding apps, namely PRESS and CCF, as follows.

1) For a given mapped trajectory segment, with PRESS, the edge sequence between every pair of two consecutive edges in the compressed trajectory follows the shortest path exactly. For the given trajectory with n edges, PRESS needs to calculate the shortest path for $n-2$ times. Readers can refer to [22] for more details.
2) For a given mapped trajectory segment, 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. Similarly, for the given trajectory with n edges, CCF needs to look up the corresponding code of out-edge from the predefined database consisting of thousands of items for at most $n-2$ times. Readers can refer to [14] for more details.

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 (1)

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

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 (2)

$$\text{cr} = \frac{N_{\text{total}} \times c_1}{\lceil N_{\text{total}}/l \rceil \times c_2 + N_{\text{edge}} \times c_3} \tag{2}$$

[5]www.openstreetmap.org

where N_{total} refers to the number of total GPS points that the local computing center received and processed; $\lceil N_{\text{total}}/l \rceil$ 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, and the occupies app size are also used to reflect its efficiency.

B. Effectiveness Study

There are important user-specified parameters (i.e., k in the trajectory mapping and l in the trajectory compression) that would impact the system performance. As this article mainly focuses on the real-system implementation and deployment, here we just fix the two parameters (i.e., $k = 6$ and $l = 10$). Readers can refer to our previous studies appeared in [6] for more evaluation results, including parameter sensitivity results, the comparison results to different baselines, and so on.

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. 6. 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. Furthermore, 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

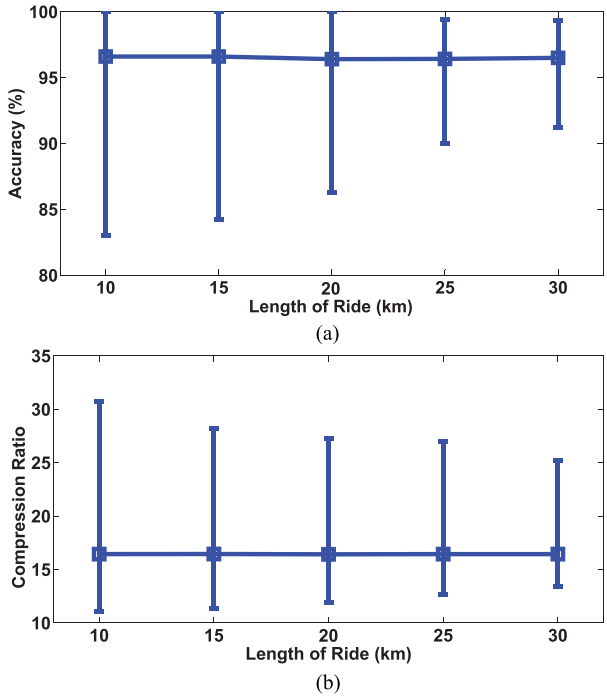


Fig. 6. Results of acc and cr for rides belonging to different groups. (a) Accuracy of trajectory mapping. (b) Compression ratio of trajectory compression.

TABLE I
COMPRESSION RATIOS OF DIFFERENT TRAJECTORY COMPRESSION SYSTEMS

System	VTracer	PRESS	CCF
cr	15.85	16.67	15.69

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 I. 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 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 [5], [28]. Under such circumstance, the edges between the origin and destination can be commonly discarded, leading to the best compression ratio.

C. Efficiency Study

1) Time Cost: Fig. 7 shows the cumulative distribution function results of the time cost at compressing each trajectory batch *with or without* the tree pruning strategy. After applying the pruning strategy, as can be seen, the time cost becomes smaller

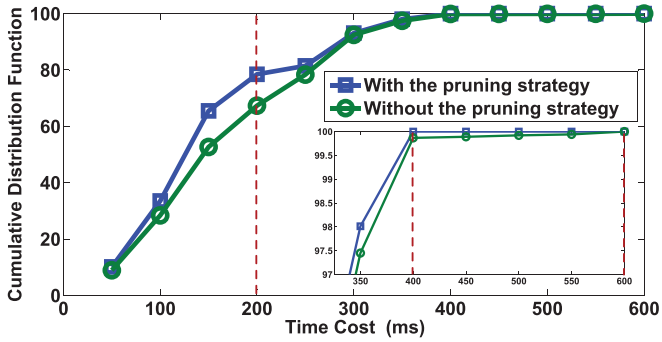


Fig. 7. Comparison results on the time cost with/without the pruning strategy.

TABLE II
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

and the system is more efficient. Thus, the effect of reducing the time cost by introducing the pruning strategy is remarkable. To be more specific, with the pruning strategy, $VTracer$ consumes less than 200 ms in most cases (around 80% of all trajectory batches), which is adequately fast for online applications. While without the pruning strategy, such percentage is less than 65%. From the figure, we can also see that the maximum time cost for $VTracer$ before and after the pruning strategy is 600 and 400 ms, respectively. As the sampling time interval is 6 s, the system can respond timely (i.e., accomplish compressing a trajectory batch before receiving the newly generated data point) no matter whether the pruning strategy is applied or not. Therefore, there is still significant room for $VTracer$ to online compress much denser GPS trajectories.

Table II 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 I and II.

2) Energy Consumption: In this article, 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 every an hour. Fig. 8 plots two curves of energy consumption

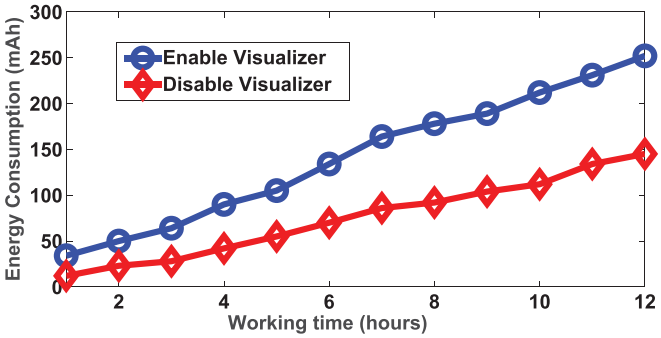


Fig. 8. Energy consumption comparison results of $TrajCompressor$ app with respect to the working time when the visualizer is disabled and enabled.

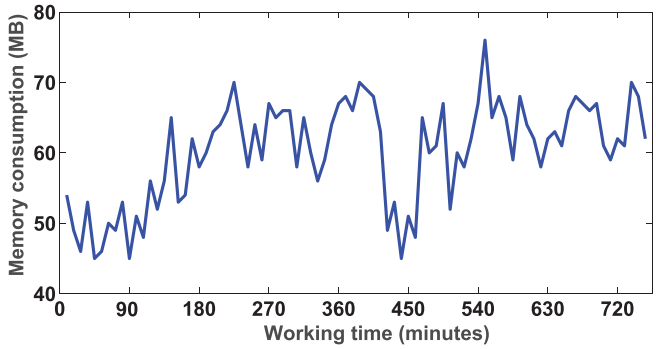


Fig. 9. Memory consumption of $TrajCompressor$ app with respect to the working time.

TABLE III
ENERGY CONSUMPTION OF DIFFERENT TRAJECTORY COMPRESSION APPS

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

TABLE IV
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

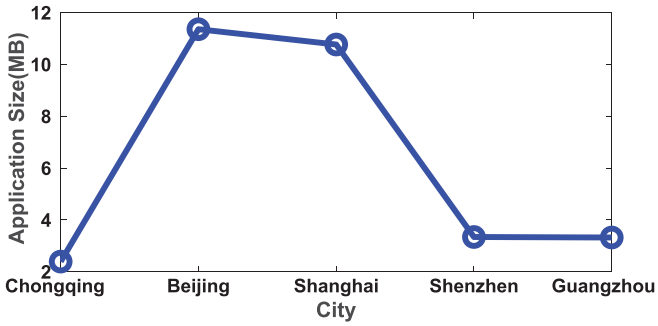


Fig. 10. App size of $TrajCompressor$ in different cities.

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 III. 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.

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. Fig. 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 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 IV. 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.

4) App Size: We also show interests at the app size of $TrajCompressor$ since users are more likely to accept and download the applications in a small size. In this article, we investigate the app size when it is deployed in five different cities in the mainland, China (i.e., Chongqing, Beijing, Shanghai, Shenzhen, and Guangzhou). Fig. 10 shows the results of app size for the five cities. As can be seen, the app size in the city of Chongqing is the smallest, occupying only 2.39 MB in space. The app size in the city of Beijing is the biggest, with a value close to 12 MB. The reason why different cities have different app sizes is because that the size of the road network in different cities is also different. The road network is essential and should be downloaded before initiating the system. It is true that the app size accumulates if one intends to support more cities, just similar to some commercial GPS navigation apps. In summary, the app size is still small and can be acceptable in real deployment.

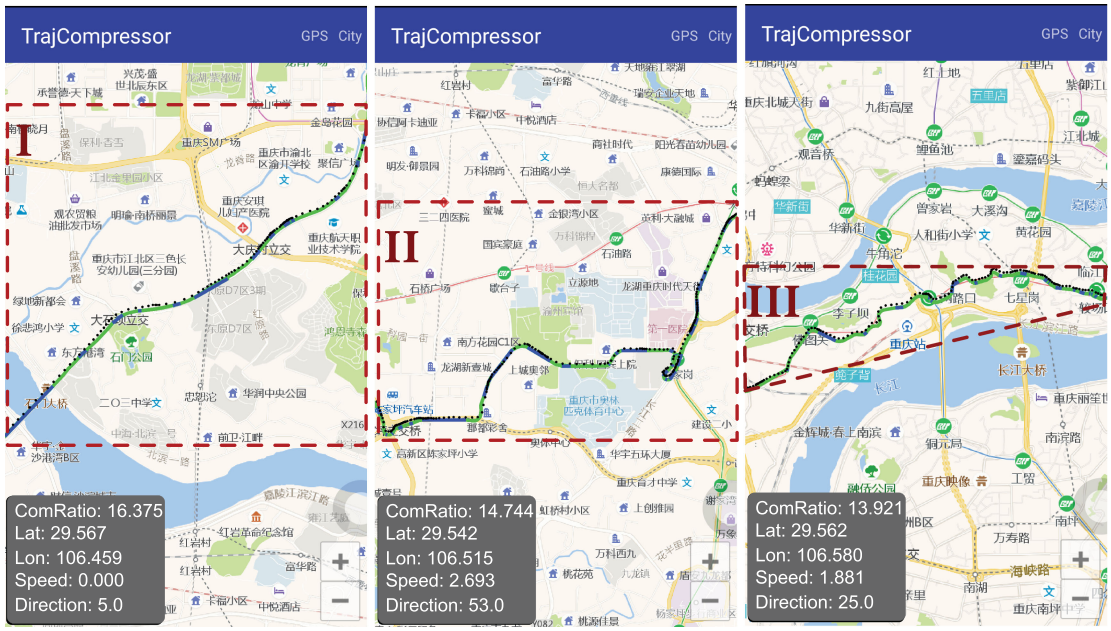


Fig. 11. Performance results of three trajectories (best viewed in an enlarged digital version).

TABLE V
SIZE OF DIFFERENT TRAJECTORY COMPRESSION APPS

App	TrajCompressor	PRESS	CCF
App size (MB)	2.39	2.37	5.31

Similar to the other studies, we compare the size of the three apps deployed in the city of Chongqing, with the results are shown in Table V. As can be seen, $TrajCompressor$ and PRESS have almost the same size, however, the size of CCF is much larger, because compared with the other two apps, CCF stores the predefined database. In more detail, every out-edge at each intersection is encoded in the clockwise order and saved in a database in advance to ensure a timely response.

TABLE VI
STATISTICAL INFORMATION ABOUT THE THREE REGIONS

Region	I	II	III
Area	7.23	7.12	4.63
# Edges	1578	2339	2834
# Nodes	1504	2056	2687
Edge density	218.26	336.94	612.10

TABLE VII
RESULTS UNDER DIFFERENT ROAD NETWORK DENSITIES

Case	Dist.(km)	Accuracy(%)	Compression Ratio	Aver. Time (ms)
I	4.6	100.00	16.375	63.23
II	7.5	96.12	14.744	179.42
III	8.1	94.17	13.921	263.05

D. Case Study

We dedicatedly choose three samples of GPS trajectory data generated by three trips when driving in three regions for our case study. Fig. 11 illustrates the specific three routes and the corresponding regions on top of the AMap. The density of the road network is quite different in different regions. For instance, the road network in Region III (downtown area in Chongqing City) is almost 2.8 times and 1.5 times denser than that in Regions I and II, respectively. More statistics about the three regions are also provided in Table VI. Note that the edge density of a region is obtained by the ratio of the number of total edges in the region to the area of a region. The unit of area is square kilometers.

The performance results of $VTracer$ in compressing these three routes are also shown in Fig. 11. Black dots in the figure refer to the raw GPS points; green solid lines refer to the mapped trajectories; blue solid line segments refer to the retained edges after trajectory compression. In each figure, some additional information is also displayed at its left bottom part, including the current compression ratio, vehicle location, speed, and heading direction. We also show the detailed performance metrics for three cases in Table VII, including the route length, the accuracy of trajectory mapping, the compression ratio of trajectory compression, as well as the average time cost. As can be seen from the table, $VTracer$ achieves slightly better performance and consumes less time in the region with the sparse road network. For instance, the accuracy of trajectory mapping is perfectly equal to 1 in Case I. The compression ratio is also the highest while the time cost is the smallest among all three cases. This is probably due to that, first, the simple structure of the road network makes the map-matching easier and more accurate, and second, drivers go more straight (i.e., make fewer turns) in a region with a sparse road network, since there are fewer intersections.

VII. Conclusion

In this article, we present a novel system called $VTracer$, with the objective of sending less data 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 improve the compression performance further by taking more actions, including compression enhancement, more advanced edge encoding methods. We also intend to investigate the trajectory compression in the temporal dimension.

Acknowledgment

The authors would like to thank J. Zhao, who helped implement CCF and PRESS algorithms on the Android OS platform.

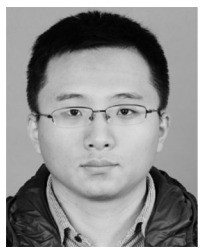
References

[1] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.
[2] H. Abid, T. C. Chung, S. Lee, and S. Qaisar, "Performance analysis of LTE smartphones-based vehicle-to-infrastrcuture communication," in *Proc. 9th Int. Conf. Ubiquitous Intell. Comput.*, 2012, pp. 72–78.
[3] G. Araniti, C. Campolo, M. Condoluci, A. Iera, and A. Molinaro, "LTE for vehicular networking: A survey.," *IEEE Commun. Mag.*, vol. 51, no. 5, pp. 148–157, May 2013.
[4] M. Bierlaire, J. Chen, and J. Newman, "A probabilistic map matching method for smartphone GPS data," *Transp. Res. Part C, Emerg. Technol.*, vol. 26, pp. 78–98, 2013.
[5] 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.*, vol. 46, no. 2, 2013, Art. no. 17.
[6] 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.
[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, Oct. 2018.
[8] C. Chen *et al.*, "CrowdDeliver: Planning city-wide package delivery paths leveraging the crowd of taxis," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 6, pp. 1478–1496, Jun. 2017.
[9] Z. Deng, W. Han, L. Wang, R. Ranjan, A. Y. Zomaya, and W. Jie, "An efficient online direction-preserving compression approach for trajectory streaming data," *Future Gener. Comput. Syst.*, vol. 68, pp. 150–162, 2017.
[10] 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.*, 2017, pp. 207–216.
[11] 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," *Cartographica: Int. J. Geogr. Inf. Geovis.*, vol. 10, no. 2, pp. 112–122, 1973.
[12] J. S. Greenfeld, "Matching GPS observations to locations on a digital map," in *Proc. 81th Annu. Meeting Transp. Res. Board*, 2002, vol. 1, pp. 164–173.
[13] Y. Ji, H. Liu, X. Liu, Y. Ding, and W. Luo, "A comparison of road-network-constrained trajectory compression methods," in *Proc. IEEE Conf. Parallel Distrib. Syst.*, 2016, pp. 256–263.
[14] 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*, 2016, pp. 472–481.
[15] G. Kellaris, N. Pelekis, and Y. Theodoridis, "Map-matched trajectory compression," *J. Syst. Softw.*, vol. 86, no. 6, pp. 1566–1579, 2013.
[16] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, and R. Jurdak, "Bounded quadrant system: Error-bounded trajectory compression on the go," in *Proc. IEEE Int. Conf. Data Eng.*, 2015, pp. 987–998.
[17] J. Liu *et al.* "A novel framework for online amnesic trajectory compression in resource-constrained environments," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 11, pp. 2827–2841, Nov. 2016.
[18] 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. Geogr. Inf. Syst.*, 2009, pp. 352–361.
[19] R. Mohamed, H. Aly, and M. Youssef, "Accurate real-time map matching for challenging environments," *IEEE Trans. Intell. Transp. Syst.*, vol. 18, no. 4, pp. 847–857, Apr. 2017.
[20] M. Satyanarayanan, "The emergence of edge computing," *Computer*, vol. 50, no. 1, pp. 30–39, 2017.
[21] 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, Oct. 2016.
[22] R. Song, W. Sun, B. Zheng, and Y. Zheng, "Press: A novel framework of trajectory compression in road networks," in *Proc. VLDB Endowment*, 2014, pp. 661–672.
[23] T. Taleb, S. Dutta, A. Ksentini, M. Iqbal, and H. Flinck, "Mobile edge computing potential in making cities smarter," *IEEE Commun. Mag.*, vol. 55, no. 3, pp. 38–43, Mar. 2017.
[24] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 54–61, Jan. 2017.
[25] C. Wang, F. R. Yu, C. Liang, Q. Chen, and L. Tang, "Joint computation offloading and interference management in wireless cellular networks with mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 66, no. 8, pp. 7432–7445, Aug. 2017.
[26] M. Wang, H. Shan, R. Lu, R. Zhang, X. Shen, and F. Bai, "Real-time path planning based on hybrid-vanet-enhanced transportation system," *IEEE Trans. Veh. Technol.*, vol. 64, no. 5, pp. 1664–1678, May 2015.
[27] S. Yi, C. Li, and Q. Li, "A survey of fog computing: Concepts, applications and issues," in *Proc. Workshop Mobile Big Data*, 2015, pp. 37–42.
[28] D. Zhang *et al.* "Understanding taxi service strategies from taxi GPS traces," *IEEE Trans. Intell. Transp. Syst.*, vol. 16, no. 1, pp. 123–135, Feb. 2015.
[29] Y. Zhang, B. Song, X. Du, and M. Guizani, "Vehicle tracking using surveillance with multimodal data fusion," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 7, pp. 2353–2361, Jul. 2018.
[30] 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.*, to be published, doi: 10.1109/TKDE.2019.2914449.

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 in computer science from Pierre and Marie Curie University and Institut Mines-Télécom/Télécom Sud-Paris, France in 2014.

He is currently a Full Professor with the College of Computer Science, Chongqing University, Chongqing, China. He has authored or coauthored more than 80 papers including 20 ACM/IEEE Transactions. His research interests include pervasive computing, mobile computing, urban logistics, data mining from large-scale taxi GPS trajectory data, and big data analytics for smart cities.

Dr. Chen work's on taxi trajectory data mining was featured by IEEE SPECTRUMin 2011 and 2016, respectively. He was also the winner of the Best Paper Runner-Up Award at MobiQuitous 2011.



Yan Ding received the B.Sc degree from the College of Mechanical Engineering, Chongqing University, Chongqing, China, in 2016 and the master's degree in computer science from the College of Computer Science, Chongqing University, Chongqing, China, in 2019.

His research interests include route planning, map matching, and spatial-temporal trajectory compression.

Zhu Wang received the Ph.D. degree in computer science from Northwestern Polytechnical University, Xi'an, China, in 2013.

He is currently an Associate Professor of computer science with the Northwestern Polytechnical University, Xi'an, China. His research interests include pervasive computing, mobile social network analysis, and mobile healthcare.

Junfeng Zhao received her Ph.D. degree in computer science from Peking University, Beijing, in 2005. She is currently an Associate Professor with Peking University, Beijing, China. She has authored or coauthored more than 50 papers in prestigious conferences and journals, such as ICWS, UbiComp, ICSP, and so on. As a Technical Leader and Manager, she has accomplished several key national projects on software engineering and smart cities. Cooperating with major smart-city solution providing companies, her research work has been adopted in more than 20 cities in China.

Her research interests include urban data analytics, ubiquitous computing, software reuse, and online software development environment.

Bin Guo received the Ph.D. degree in computer science from Keio University, Tokyo, Japan in 2009.

He is currently a Professor with the Northwestern Polytechnical University, Xi'an, China. He was a Postdoctoral Researcher with the Institut Mines-TELECOM/TELECOM SudParis, France. His research interests include ubiquitous computing, mobile crowd sensing, and HCI.

Daqing Zhang (F'19) received the PhD degree from the University of Rome "La Sapienza" and University of L'Aquila in 1996. He is currently a Professor with the Telecom SudParis, France. He has authored or coauthored more than 180 referred journal and conference papers, all his research has been motivated by practical applications in digital cities, mobile social networks, and elderly care. His research interests include large-scale data mining, urban computing, context-aware computing, and ambient assistive living.

Dr. Zhang is the Associate Editor for four journals including *ACM Transactions on Intelligent Systems and Technology*. He has been a frequent Invited Speaker in various international events on ubiquitous computing. He is a recipient of the 10 Years CoMoRea Impact Paper Award at IEEE PerCom 2013, the Best Paper Award at IEEE UIC 2015/2012, and the Best Paper Runner Up Award at Mobiquitous 2011. He is a member of the China Thousand-Talent Program.