TrajCompressor: An Online Map-matching-based Trajectory Compression Framework Leveraging Vehicle Heading Direction and Change

Chao Chen, Yan Ding, Xuefeng Xie, Shu Zhang, Zhu Wang, and Liang Feng

***Abstract*— Massive and redundant vehicle trajectory data are continuously sent to the data center via vehicle-mounted GPS devices, causing a number of sustainable issues, such as storage, communication, and computation. Online trajectory compression becomes a promising way to alleviate these issues. In this paper, we present an online trajectory compression framework running under the mobile environment. The framework consists of two phases, i.e., online trajectory mapping and trajectory compression. In the phase of online trajectory mapping, we develop a light-weighted yet efficient map matcher, namely, Spatial-Directional Matching (SD-Matching), to align the noisy and sparse GPS points upon the underlying road network, which fully explores the usage of vehicle heading direction collected from the GPS trajectory data. In the phase of online trajectory compression, we propose a novel compressor based on the heading change at intersections, namely, Heading Change Compression (HCC), aiming at finding a concise and compact trajectory representation. Finally, we conduct experiments to evaluate the effectiveness and efficiency of the proposed framework using real-world datasets in the city of Beijing, China. We further deploy the system in the real world in the city of Chongqing, China. The experimental results demonstrate that: 1) the SD-Matching algorithm achieves a higher mean accuracy but consumes less time than the state-of-the-art algorithm, namely, Spatial-Temporal Matching (ST-Matching) and 2) the HCC algorithm also outperforms baselines in trading-off compression ratio and computation time.**

Manuscript received January 4, 2018; revised April 9, 2018, July 29, 2018, October 14, 2018, December 4, 2018 and February 2, 2019; accepted April 8, 2019. Date of publication April 23, 2019; date of current version May 1, 2020. This work was supported in part by the National Key Research and Development Project of China under Grant 2017YFB1002000, in part by the National Science Foundation of China under Grant 61872050, Grant 61602067, and Grant 71601024, in part by the Fundamental Research Funds for the Central Universities under Grant 2018cdqyjsj0024, and Grant 2019cdxyjsj0022, in part by the Frontier Interdisciplinary Research Funds for the Central Universities under Grant 106112017cdjqj188828, in part by the Chongqing Basic and Frontier Research Program under Grant cstc2018jcyjAX0551, and in part by the Ministry of Education in China Humanities and Social Sciences Youth Foundation under Grant 16yjc630169. The Associate Editor for this paper was J. Sanchez-Medina. *(Chao Chen and Yan Ding contributed equally to this work.) (Corresponding authors: Chao Chen; Xuefeng Xie.)*

C. Chen, Y. Ding, X. Xie, and L. Feng 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: cschaochen@cqu.edu.cn; snowflakex90@gmail.com).

S. Zhang is with the School of Economics and Business Administration, Chongqing University, Chongqing 400044, China (e-mail: zhangshu@cqu.edu.cn).

Z. Wang is with the Department of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China (e-mail: transitwang@gmail.com).

Digital Object Identifier 10.1109/TITS.2019.2910591

***Index Terms*— Vehicle trajectory compression, map-matching, heading direction, heading change, mobile environment.**

I. Introduction

MASSIVE trajectory data of moving objects is being accumulated and recorded at an unprecedented speed and scale, as a result of the proliferation of GPS-enabled devices and the maturity of Mobile Internet in recent years. Vehicle trajectory data is a typical example but with a particularly spatial constraint, because vehicles can only move on the underlying road networks. Currently, the locations of moving vehicles are usually sampled and reported to the data center at a constant time interval, which is generally redundant and far more necessary for many applications in real life, causing communication overhead, storing and computing issues [23], [38], [46], [49]. Even worse, the communication and storing of trajectory data are typically costly, because an increasing number of public and private vehicles have been equipped with GPS devices and are continuously sending their trajectory data to the data center [19], [20]. What is more, massive data also hinders computing tasks, including visualization, pattern mining and so on.

Reducing the size of trajectory data *before sending it to the data center* is one of the promising ways to alleviate the above-mentioned issues [11], [34], [49]. The most intuitive way is to collect less trajectory data, by reporting the vehicle's locations less frequently. However, such method is problematic since the collected data may be too sparse to infer the detailed driving paths in-between, resulting in limited urban services [35], [54]. In contrast, *online trajectory compression* tries to address the issues from the perspective of seeking trajectory representation, which is a common solution but never well-addressed so far [38], [49]. Depending on different applications, many trajectory compression methods have been proposed, among which the most well-known are the Douglas-Peucker (DP) algorithm and its variants [13], [32], [37]. Their core idea is to simplify trajectory by keeping some important GPS points, according to the contribution to the trajectory shape. However, the importance of a GPS point may be wrongly evaluated due to the positioning error of GPS devices. With the widely available road network data, vehicle trajectory is often mapped to a path in the road network (i.e., trajectory

mapping or map-matching) in advance [23], [26], which is proved to: 1) have the potential to reduce the side effects caused by the location error significantly; 2) generate a more *natural and semantical* trajectory representation. Therefore, in general, map-matching based trajectory compression methods outperform non-map-matching based algorithms. Unfortunately, the task of trajectory mapping is generally computation-intensive and resource-consuming. The computation capability of GPS devices mounted on moving vehicles is quite limited and they cannot afford such heavy tasks. Thus, *developing a light-weighted yet efficient map-matching algorithm which is workable under the mobile environment is our first objective.*

The mapped trajectory is returned by the map-matching, which is usually represented by a sequence of connected edges in the road network [8], [9], [55]. However, such representation is redundant and can be further reduced, motivating the research of *trajectory compression*. In essence, trajectory compression aims at finding a *compact and concise* trajectory representation. More precisely, the objective of trajectory compression is to remove some edges in the trajectory that are unnecessary and can be recovered using the maintained edges. A representative algorithm is PRESS [46], which exploits the idea of shortest-path to accomplish trajectory compression. In more detail, given a mapped sub-trajectory from edges e_i to e_j, if in-between edge sequence follows the shortest path exactly from e_i to e_j, the sub-trajectory can be represented by $\langle e_i, e_j \rangle$. The rationale behind is that the in-between edge sequence can be easily inferred (or recovered) by using the edge pair only. As drivers tend to take shortest paths in many cases, such compression can effectively reduce the number of edges we have to maintain for each trajectory. However, PRESS needs to compute and store all shortest paths between any two pair of edges in the road network, which is *infeasible* under the mobile environment. For instance, the storage space of all shortest paths is over 100 GB for the city of Beijing. Therefore, *developing a cost-effective online trajectory compression algorithm on top of the mapped trajectory is another objective.*

With the above-mentioned research objectives and challenges, the **main contributions** of the paper are:

- To alleviate the issue caused by the limited computing capacity of GPS devices, we migrate the computation burden to the mobile phones of drivers since mobile phones are almost idle while driving and are more capable of the computation-intensive tasks.
- We develop a novel framework of trajectory compression called TrajCompressor, which contains two phases, one is trajectory matching and the other is trajectory compression. In both phases, we systematically leverage a new dimension of collected GPS trajectory data (i.e., vehicle *heading direction*) that is under-explored in prior work on trajectory mapping and compression [43], [48], [60]. In the phase of trajectory matching, we propose a three-stage online map-matcher, namely SD-Matching that fully exploits the heading direction in all stages. In the phase of trajectory compression, we propose a new trajectory compressor, namely HCC that uses another new information derived from the heading direction (i.e., *heading change* at intersections) in a smart way.
- We 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 TrajCompressor. Experimental results show that TrajCompressor achieves a high compression ratio and can respond in real time. More specifically, 1) SD-Matching achieves a higher mean accuracy yet consumes less time than the state-of-the-art map matcher (i.e., ST-Matching); 2) HCC also achieves a much better performance than baselines in balancing compression ratio and computation time.

The rest of the paper is organized as follows: In Section II, we review the related work and show how this paper differs from prior research. In Section III, we introduce some basic concepts and provide an overview of the system. We elaborate the algorithm details on the trajectory mapping and the trajectory compression using heading direction and heading change in Section IV and V, respectively. We evaluate the performance of the proposed framework in Section VI. Finally, we conclude the paper and discuss the future research directions in Section VII.

II. Related Work

In this section, we briefly review the related work which can be grouped into two categories, i.e., trajectory mapping and trajectory compression, respectively.

A. Trajectory Mapping

The general procedure of online map-matching consists of *finding the true position, inferring, and refining the true travelling paths*. Two nice survey papers on map-matching can be found in [29], [43]. [58] summarizes several map-matching algorithms in their overview of trajectory data mining, since the map-matching is known as its prerequisite. Earlier research work mainly focused on finding the "true" position for individual GPS point. For instance, [2], [3], [17], [52] make use of geometric analysis based on the geometric information such as the distance of 'point-to-curve' or 'point-to-shape points'. As a matter of fact, heading direction demonstrates the potential to increase the accuracy of true position locating. Thus, we attempt to investigate how to use the heading direction to enhance the mapping performance in this study.

However, the above-mentioned work only addresses the measurement errors of GPS devices. More recent work on map-matching mainly focused on filling the distance gap between sampled GPS points by reconstructing the vehicle travelling path. To be more specific, the two roads (edges) where two consecutive GPS points are located are generally not connected and the gap between them can be large. In this regard, this kind of work mainly targets at addressing the issues of *sparseness and uncertainty* [57]. To infer the path accurately, additional information such as the road network topology (e.g., link connectivity and contiguity), road attributes (e.g., number of lanes, speed limits) and motion laws are used [24], [35], [54]. Furthermore, some probabilistic

algorithms (e.g., Hidden Markov Model-based) and more advanced algorithms are also proposed to increase the matching quality [4], [5], [16], [28], [39], [41], [57], [60]. To infer the true path for a considered sequence of GPS points, several travelling paths between pairs of two consecutive GPS points are inferred first, followed by the path refining via the neighboring GPS points and road network topology comprehensively. In the process of path-inferring for each pair of consecutive GPS points, Dijkstra and A* algorithms are commonly used [39]. The run-time is extremely long since the shortest-path computation is required between every combination of candidate edges of two consecutive GPS points [51]. *In this paper, based on the collected heading direction, we propose a heuristic to explore its usability in both path finding and refining, focusing on overcoming the run-time bottleneck in the shortest-path computation.*

B. Trajectory Compression

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

1) Line-Simplification-Based Compression: The core idea of line-simplification-based algorithms is to maintain some important GPS points in the original trajectory, according to their contribution to the trajectory shape [13], [27], [32], [33], [36]–[38]. 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 used. The GPS point is claimed important if the distance is bigger than the user-specified error bound. Hence, the error bound value has a significant impact at the number of retained GPS points and the compression ratio. The DP and its variants are the most well-known representatives [13]. 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 that align the GPS points to the underlying road network before trajectory compression are proposed.

2) Map-Matching-Based Compression: This group of compression methods generally maps raw trajectory data onto the road network in advance [21]–[23], [25], [26], [40], [46]. The utilization of road network endows the trajectory with more correct and semantical information, which further benefits the trajectory compression. Some recent work also introduces the trajectory prediction in the compression. In [45], based on the historical trajectory data, if the actual trajectory is not consistent with the results predicted by the Markov model, the trajectory would be retained, otherwise, it would be discarded. In the overview of trajectory data mining, [15], [58] briefly summarize some popular trajectory compression algorithms. Here, we select three well-known compressors to review, i.e., MMTC [26], PRESS [46], CCF [23].

As we know, there may be several paths between two points in the road network, each of which usually consists of various number of edges, i.e, some paths are just with 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, different from MMTC, PRESS, CCF and HCC are spatial-lossless.

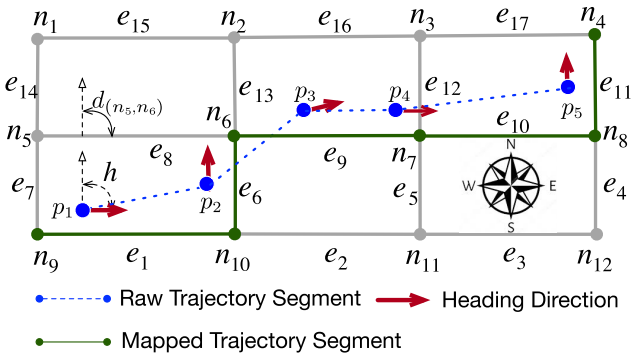


Fig. 1. Illustration of main concepts used in this paper.

III. Preliminary

A. Basic Concepts

Definition 1 (Edge Direction): Each edge e_i usually has two edge directions, i.e., $d(n_h, n_t)$ and $d(n_t, n_h)$ respectively. $d(n_h, n_t)$ is defined as the edge direction from node n_h to n_t, which can be easily calculated based on the longitudes and latitudes of nodes, using north direction as a basis. For instance, $d(n_5, n_6)$ in Fig. 1 is the edge direction of e_8 from n_5 to n_6.

Definition 2 (Heading Direction): Heading direction h ($0° < h \leq 360°$) is the azimuth that the vehicle heads at a sampling position. This information can be accessible directly from the trajectory data. For example, the vehicle's heading direction at point p_1 is h_1, as shown in Fig. 1.

Definition 3 (Raw Trajectory Segment): A raw trajectory segment τ is a sequence of GPS points, represented by $\tau = \langle p_i, p_{i+1}, \cdots, p_{i+l-1} \rangle$. A GPS point p_i records the spatio-temporal information of a vehicle, which consists of a timestamp, a geospatial coordinate, an instantaneous speed, and a heading direction, denoted by $p_i = (t_i, lat_i, lon_i, v_i, h_i)$. Parameter l controls the length of

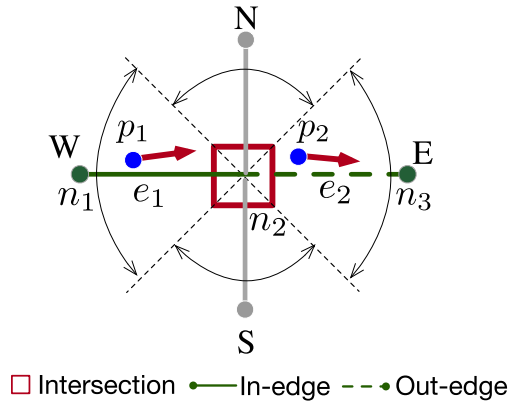


Fig. 2. Illustration of heading change, in- and out-edges.

the segment. A raw trajectory stream T consists of an unbounded set of raw trajectory segments τ, denoted by $T = \langle \tau_1, \tau_2, \cdots \rangle$.

Definition 4 (Mapped Trajectory Segment): Given a raw trajectory segment, its mapped segment τ_m is the real path that the vehicle travels in the road network. τ_m is denoted by $\langle e_i, e_{i+1}, \cdots, e_n \rangle$, where each pair of two consecutive edges are connected.[1] *A mapped trajectory stream $T_m = \langle \tau_{m1}, \tau_{m2}, \cdots \rangle$ is an unbounded set of mapped trajectory segments τ_m.*

Definition 5 (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. τ_c is a subset of τ_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 τ_c and their starting times, denoted by $T_c = \langle \{t_1, \tau_{c1}\}, \{t_2, \tau_{c2}\}, \cdots \rangle$.

Definition 6 (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. 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 7 (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. 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 such manner, heading changes of mapped trajectory segments can be also estimated, since the heading direction information is not saved any more in mapped trajectory segments.

[1]For consistence, in this paper, we also define the length of mapped trajectory segment as the length of the corresponding raw trajectory segment.

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

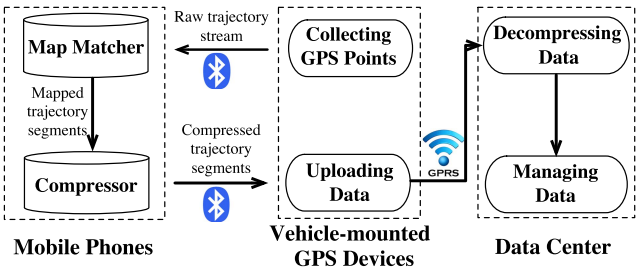


Fig. 3. System overview of TrajCompressor framework.

B. System Overview

The framework of TrajCompressor consists of three major units, namely *vehicle-mounted GPS devices*, *mobile phones* and *data center*, as shown in Fig. 3. The *vehicle-mounted GPS devices* serve two main functions. The first function is that the devices continually collect the GPS points at a constant rate, then transmit raw trajectory stream (an unbounded set of GPS points) into mobile phones via bluetooth. The other is that GPS 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*. Taking raw GPS trajectory stream as input, *Map Matcher* first splits the incoming data stream into an unbounded set of raw trajectory segments. It should be noted that there is an overlapped point between two consecutive segments, i.e, the last GPS point of the former segment is also the first GPS point of the latter segment. The segment length is controlled by a user-specified parameter (i.e., l). Then, for each raw trajectory segment, *Map Matcher* maps it into a sequence of connected edges (i.e., the mapped trajectory segment) in the road network (i.e., map-matching). Thereafter, *Compressor* compresses each mapped trajectory segment and returns the compressed representation to the GPS devices. Both map-matching and trajectory compression are implemented in mobile phones. These two phases run *sequentially* to return the compressed trajectory segments. 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. The data may be decompressed and managed in the data center for further applications, such as the common *when-and-where* query service.

IV. Phase I: A Three-Stage Map-Matching Algorithm Using Heading Direction

A. Identifying Top-k Candidate Mapped Edges

For a given GPS point p_i, we first determine a set of candidate mapped edges within the circle of radius r in the

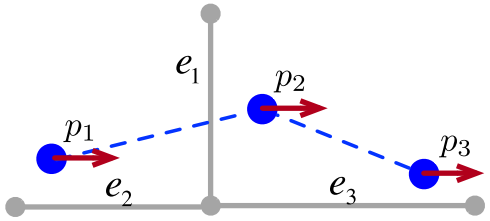


Fig. 4. An illustrative example demonstrating the utility of heading direction in map matching.

road network. To ensure that the true edge that the given GPS point located can be included in the set, we choose a slightly bigger value of r (i.e., 100 meters) than the error range of GPS devices. Note that each point p_i commonly has several candidate mapped edges, especially in the area with the dense road network. To find the truly correctly-matched edge for the point, it is necessary to reduce the number of candidate mapped edges. Here, for each candidate mapped edge, we first use the *spatial probability* and *direction probability* together to measure the probability of obtaining a correctly-matched result for the given GPS point, then identify the top-k candidate edges for further processing.

1) Spatial Probability: The spatial probability is defined as the *likelihood* that a GPS point p_i matches a candidate edge e_j based on the distance between p_i and e_j. The distance ($H_{e_j}^{p_i}$) from p_i to e_j is the minimal one among $dist(p_i, c_1), dist(p_i, c_2), dist(p_i, c_3)$, where function $dist$ returns the distance between two points; c_1 and c_2 are the two end-points of edge e_j. The point c_3 is the perpendicular projection point on edge e_j of p_i. Note that we manually set $dist(p_i, c_3)$ to $+\infty$ if c_3 is out of the range of e_j.

Generally speaking, the noise of location measurement can be reasonably modeled as a standard-normal distribution of the distance $H_{e_j}^{p_i}$ [35]. Such distribution indicates that how likely p_i can be matched to the candidate edge e_j in the road network. Thus, the spatial probability can be calculated based on Eq. 1.

$$G_1(H_{e_j}^{p_i}) = \frac{1}{\sqrt{2\pi}\sigma_1} e^{-\frac{(H_{e_j}^{p_i})^2}{2{\sigma_1}^2}} \tag{1}$$

where σ_1 is a standard deviation of the position measurement error. To estimate the value of σ_1, for a set of raw trajectories, we first obtain their corresponding mapped trajectories using popular map matching algorithms offline. For each GPS point in all raw trajectories, distance from itself to the mapped edge can be easily computed. Finally, the σ_1 value can be derived based on the distance distribution.

However, merely relying on the spatial probability might lead to the mismatched result. Fig. 4 shows such an example. Based on the distance, the GPS point p_2 should be mapped to edge e_1, which is incorrect in the real case. As can be seen, the vehicle heading direction at the point p_2 is much closer to edge e_3, which motivates us to bring in the heading direction information to avoid mismatching. More specifically, *direction probability* is introduced, detailed as follows.

2) Direction Probability: The direction probability is defined as the *likelihood* that a GPS point p_i matches a candidate edge e_j based on the angle difference between the vehicle's heading direction h_i at the point p_i and the edge direction e_j. The angle difference ($A_{e_j}^{p_i}$) between p_i and e_j is computed as $min\{|h_i - d_{th}|, |h_i - d_{ht}|\}$.

Similar to the spatial probability, the direction probability can be calculated according to Eq. 2.

$$G_2(A_{e_j}^{p_i}) = \frac{1}{\sqrt{2\pi}\sigma_2} e^{-\frac{(A_{e_j}^{p_i})^2}{2{\sigma_2}^2}} \tag{2}$$

where σ_2 is a standard deviation of direction measurement error, which can be estimated in the similar way to σ_1.

Combining the probability G_1 and G_2, we define the *comprehensive probability* G as the geometrical mean of the spatial probability and the direction probability, as shown in Eq. 3.

$$G = \sqrt{G_1(H_{e_j}^{p_i}) \times G_2(A_{e_j}^{p_i})} \tag{3}$$

As mentioned previously, the GPS point may have a large number of candidate mapped edges within the user-specified radius. Each candidate mapped edge has a value of comprehensive probability G. We can identify k candidate edges with top probability values (i.e., top-k candidate edges) for each GPS point. Note that the number of candidate edges (i.e., the parameter k) may impact the accuracy and time cost of the proposed map matching algorithm, which will be evaluated in Section VI.

B. Finding Potential Paths

The candidate edges of two consecutive GPS points p_i and p_{i+1} are generally unconnected in the road network, since the distance between them may be far. Thus, we need to find potential paths between two consecutive GPS points. For each point p_i, it has k candidate edges. It is easy to understand that for a pair of two consecutive points, it would have k^2 possible paths in total. In previous work, such a path is commonly obtained by A* or Dijkstra algorithm, which is time-consuming. In our case, the computation time issue is even more critical, because all k^2 possible paths are needed. To alleviate this issue, we propose to *make full use* of the heading direction to 1) *narrow down the searching zone* and 2) *serve as a cost-effective guider* in finding paths.

The proposed path-finding algorithm includes two steps, i.e., *determining the potential searching zone and finding paths within the potential searching zone*.

Step 1 (Determining the Potential Searching Zone): The potential searching zone is a sector (e.g., the gray area in Fig. 5), which can be well-determined as follows: 1) the apex of this sector is point p_i; 2) the radius r_{max} of the sector is calculated by $\max(v_{p_i}, v_{p_{i+1}}) \times \Delta t + c$, where v_{p_i} and $v_{p_{i+1}}$ are the speed values of the vehicle at points p_i and p_{i+1} respectively; c is constant (we set $c = 50$ meters) and Δt is the sampling time interval. Note that r_{max} indicates the maximum driving distance of the vehicle from point p_i to p_{i+1}; 3) the sector is composed of two semicircles. Each diameter of semicircles are perpendicular to the vehicle's heading direction h_i and h_{i+1} at point p_i and p_{i+1} (solid and dashed semicircles in Fig. 5), respectively.

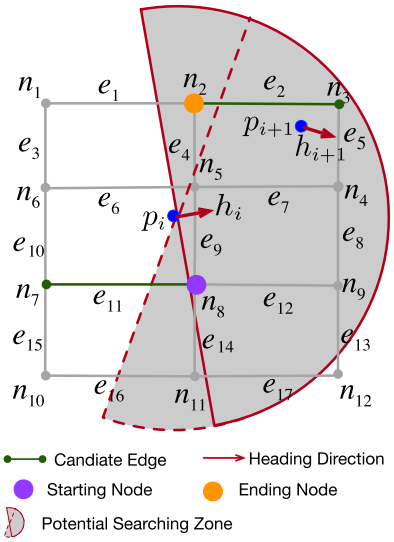


Fig. 5. An illustrative example of the proposed two-step path-finding algorithm.

Step 2 (Finding Paths Within the Potential Searching Zone): For the given GPS points p_i and p_{i+1}, we need to find k^2 possible paths corresponding to k^2 pairs of candidate edges within the potential searching zone (returned by *Step 1*). For a pair of candidate edges, according to the heading directions at p_i and p_{i+1}, it is easy to determine the starting node and the ending node of the path. For instance, for the candidate edge pair of e_{11} and e_2, we can easily obtain that n_8 and n_2 are the starting node and the ending node respectively, as shown in Fig. 5. Thus, the problem of path-finding for a pair of candidate edges is simplified to the one of finding path from a starting node to an ending node. Note that there might be just no path connecting the starting and the ending nodes that can be discovered within the searching zone.

We propose a *heading-direction-guided* algorithm that exploits the heading direction as a guider to find the path for a pair of starting and ending nodes ($\langle n_s, n_e \rangle$) efficiently. First of all, for each node within the potential searching zone, we determine its child nodes according to its position and the heading directions at the two consecutive GPS points (i.e., p_i and p_{i+1}). More specifically, for a node n_i, a node (n_c) can be its child node *if and only if* it meets the two conditions: 1) the child node n_c should connect to node n_i in the road network fragment. In addition, n_c is closer to n_e and farther from n_s, compared to n_i, as shown in Eqs. 4 and 5; 2) the direction from n_i to the child node should be in consistence with the two heading directions, mathematically, satisfying the following Eq. 6.

$$dist(n_c, n_e) < dist(n_i, n_e) \tag{4}$$

$$dist(n_c, n_s) > dist(n_i, n_s) \tag{5}$$

$$\min(|h_i - d(n_i, n_c)|, |h_{i+1} - d(n_i, n_c)|) \leq 90^\circ \tag{6}$$

Secondly, we build up *a tree* based on the determined node relationships, with the starting node n_s as the *root node*. If the constructed tree does not contain n_e, we can simply report that there should be no path connecting this pair of nodes. Otherwise, on top of the tree, we can apply the *Depth-First-Search (DFS)* algorithm to find the path [50]. Once a path from the starting node to the ending node has been found, the search would be terminated for this pair. The whole procedure would be terminated until all pairs of nodes are checked.

C. Refining Paths

Between two consecutive GPS points, with the previous two steps, we could obtain at most k^2 possible paths. Thus, for a raw trajectory segment with l GPS points, ideally there would be totally $k^{2(l-1)}$ possible paths connecting them. However, the actual number is much smaller since 1) there are a much smaller number of paths ($\ll k^2$) for two consecutive GPS points (i.e., there would be no path returning for some node pairs); 2) not all paths of different consecutive GPS points can be connected in the road network, and path refining based on the topology of the road network is necessary. Moreover, among all possible paths, each of them has different probabilities to be the real mapped trajectory, as each candidate edge has different probabilities to be the correctly mapped one for the GPS point. Thus, we further refine the path based on the possibility of a path (τ_{mi}) being the real mapped trajectory. The possibility is estimated by summarizing all the comprehensive probabilities of GPS points being correctly mapped to the candidate edge, as shown in Eq. 7.

$$P_{\tau_{mi}} = \sum_{i=1}^{l} G(p_i, e_j^{p_i}) \tag{7}$$

where $G(p_i, e_j^{p_i})$ refers to the comprehensive probability of p_i being correctly mapped to the edge e_j, as defined in Eq. 3. Note that one possible path for a given raw trajectory segment $\langle p_1, p_2, \cdots, p_\delta \rangle$ can be denoted by $\tau_{mi} = \langle e_j^{p_1} \rightsquigarrow e_j^{p_2} \cdots \rightsquigarrow e_j^{p_\delta} \rangle$, where $e_j^{p_i}$ refers to the jth candidate edge of p_i; $e_j^{p_i} \rightsquigarrow e_j^{p_{i+1}}$ refers to the path from $e_j^{p_i}$ to $e_j^{p_{i+1}}$.

V. Phase 2: A Trajectory Compression Algorithm Using Heading Change

A. 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).

In the case of recommending driving directions, to avoid distracting drivers, navigation systems actually provide a more concise trajectory representation, i.e., using heading changes at intersections. Inspired by the idea, authors in [23] propose a Clockwise Compression Framework (CCF). In CCF, from the origin to the destination, the trajectory is represented by the sequence of out-edge at each by-passing intersection. The out-edge is encoded with a unique code based on the ID of the intersection and the heading change. The number of elements in the representation is just the number of by-passing

intersections (excluding the starting and ending edges). The compressed trajectory with CCF occupies less space since the number of intersections is smaller than the number of edges. In daily life, we are experienced "going straight" more often at intersections during driving. Under such circumstance, out-edges with small heading change is not necessary to be maintained. Therefore, 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. We name this compression algorithm as Heading Changes Compression (HCC for short) since it is based on heading changes.

B. Heading Changes Compression (HCC)

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 1 summarizes the whole procedure of the proposed HCC algorithm. Note that the mapped trajectory segment (τ_m) is the basic processing unit of HCC.

Algorithm 1 $HCC(\tau_m, G(N, E))$

1: $\tau_c = e_1$;
2: **for** $i == 2$ to $|\tau_m| - 1$ **do**
3: $s = identifyNode(e_i, e_{i+1})$;
4: **if** $isIntersection(s, G(N, E))$ **then**
5: **if** $\sim isGoStraight(e_i, e_{i+1})$ **then**
6: $\tau_c = \tau_c \cup e_{i+1}$;
7: **end if**
8: **end if**
9: **end for**
10: $\tau_c = \tau_c \cup e_{|\tau_m|}$;

Line 1 in Algorithm 1 refers to the initialization of the compressed trajectory. To be more specific, HCC enrolls the first edge e_1 of the input trajectory into the compressed trajectory τ_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 τ_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 τ_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 $\mathcal{O}(|\tau_m|)$, where $|\tau_m|$ is the number of edges in the input mapped trajectory τ_m.

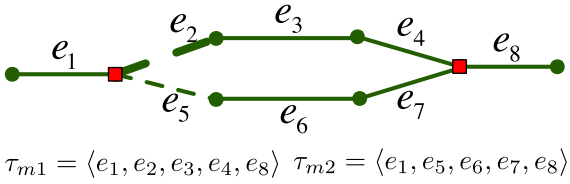


Fig. 6. Illustrative example of trajectory ambiguity.

However, HCC may cause *trajectory ambiguity*, due to the complexity of the road network and the coarse granularity of the defined heading change category (refer to **Definition** 6). For instance, 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. 6. 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.

To address the issue, an intuitive idea is 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 the Fig. 6 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: The idea is inspired by the daily experience that drivers may prefer to select some out-edge than others when traversing the intersections, just similar to the observation that drivers are inclined to choose the shortest path from the origin to the destination. Therefore, 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. 6 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 τ_{m1} during compression. Bringing in the idea of FEC, the compressed trajectories for τ_{m1} and τ_{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.

C. 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. 6 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 [44]. 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 [6], [18], [30]. 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 [44], [49].

VI. Evaluation

In this section, based on the GPS trajectory data collected from taxis in the real world, we conduct extensive experiments to show the effectiveness and efficiency of the proposed TrajCompressor framework. More specifically, the performance of the SD-Matching algorithm in terms of matching accuracy and computation time, and HCC algorithm in terms of compression ratio and computation time are evaluated. In addition, we would like to examine the boundary choice of the length of trajectory segment in the proposed framework. Finally, we also test the performance of HCC algorithm 1) under various densities of the road network, 2) on processing GPS trajectories with various sampling rates, and 3) on the quality of response for the *when-and-where* query.

A. Baselines

Since the proposed framework contains two phases, in each phase, we compare the proposed algorithm to the state-of-art, with the details as follows.

Baseline for SD-Matching Algorithm. As a comparison, ST-Matching algorithm, which is well-recognized as the state-of-art map-matching algorithm and the most popular competitor [35], [54], is used as the baseline algorithm. It generally follows a same three-step map-matching procedure as our proposed SD-Matching algorithm, but it differs in the following two main aspects:

- For a given GPS point, it identifies the candidate edges based on spatial information merely, while our method considers both the spatial and *heading direction* information collectively. More precisely, when calculating the probability of a candidate edge of being the correctly-matched edge for the given GPS point, it uses both spatial and temporal information, while our method uses both spatial and heading direction information.
- When searching the path between two consecutive GPS points, it adopts A* algorithm. In contrast, we propose a novel path-finding algorithm that fully uses the heading direction information again.

Baselines for HCC Algorithm: Three baselines are used to compare, i.e., PRESS [46], CCF [23] and DP algorithms [13], with their details as follows.

- 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. Readers can refer to [46] for the details.
- 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.
- 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. Thus, the compression ratio is largely affected by the value of user-specified error bound.

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.*

B. Experimental Setup

1) Data Preparation: Two different kinds of datasets from the city of Beijing, China, including one road network dataset and two taxi GPS trajectory datasets, are used in the experiments. The basic dataset is the road network, which can be freely downloaded and extracted from OpenStreetMap.[3] In total, it contains 141,735 nodes and 157,479 edges. What is more, OpenStreetMap also provides some additional attributes of edges, including speed limits, number of lanes, number of traffic lights and so on [7], [10], [12].

The second and third datasets are the GPS trajectory data (labeled as *Taxi_I* and *Taxi_II* respectively), both of which

[3] http://www.openstreetmap.org/

are generated by taxis, but in different times and scales. The sampling rate for all taxis in both datasets is identical and constant, with a value of around 6 seconds. *Taxi_I* is generated by 50 taxis on 15th September, 2015, which contains over 50,000 GPS points. It is mainly used to evaluate the performance of *map-matching algorithms* that needs the "true" taxi trajectories lying on the road network. To get such true trajectories, for each set of raw trajectory data generated by one taxi, we manually split it into a number of raw trajectory segments containing equal-sized GPS points. For each segment, with the digital map, we recruit three volunteers to identify its corresponding mapped trajectory segment on the road network manually. The human-labelled mapped trajectory segments can be viewed as the ground truth for the raw trajectory segments. The scale cannot be large since the human-labeling work is both labor- and time-intensive. *Taxi_II* is much larger, which is generated by 595 taxis in one week (from 15th to 21th September, 2015). It is used to test the performance of *trajectory compression algorithms*. In terms of storage space, *Taxi_II* takes up over 1.01 GB.

2) Evaluation Metrics: For map-matching algorithms, the metric of *accuracy* is proposed to measure the accuracy of map-matching algorithms of a given raw trajectory segment (τ_i), which is defined as the ratio of *the number of GPS points that are corrected mapped* to *the number of total GPS points to be mapped in the segment*, as shown in Eq. 8.

$$acc(\tau_i) = \frac{\#correctly\ mapped\ GPS\ points}{\#GPS\ points\ to\ be\ mapped} \tag{8}$$

We simply use the mean value of the accuracy for all trajectory segments ($acc_m = \sum_{i=1}^{N} acc(\tau_i)$) to measure the overall accuracy of the map-matching algorithms. It should be mentioned that the matching accuracy is largely affected by the quality of the road network. If a road network is inaccurate (i.e., containing error nodes or edges) or incomplete (i.e., missing some roads), the matching accuracy would be certainly low regardless of the map-matching algorithms. Road network inaccuracy is associated to the scale and the representation of the real-world on a digital map. To be more specific, the currently available road network is often vectorized and represented by a node-edge structure [29], in which both node and edge may be not exactly true. More precisely, the position of node may be wrong; the width of road is ignored since the edge can only represent the centerline of the true multi-lane road. Also, such simple structure cannot well represent the curve shapes. In addition, the road network used in the paper is downloaded from OpenStreetMap, in which the road network is produced via a crowdsourced way. In such manner, volunteers may upload inaccurate road information.

To evaluate the efficiency of the map-matching algorithms, the average computation time which is the mean value of all time costs at projecting trajectory segments onto the road network is used.

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.

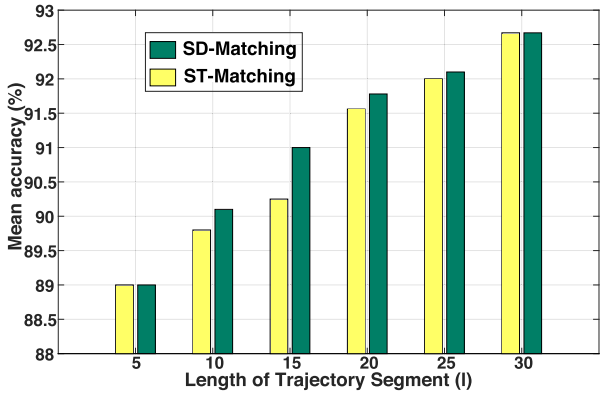


Fig. 7. Comparison results of mean accuracy for different algorithms under different ls.

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.

C. Effectiveness Study

1) SD-Matching Algorithm: There are two important user-specific parameters that can influence the mean accuracy of SD-Matching algorithm. One is the length of the trajectory segments (i.e., l), and the other is the number of identified candidate edges for each given GPS point, referred as k. We study the performance of SD-Matching algorithm on the mean accuracy under different ls and ks, respectively.

Varying l: We report the results of mean accuracy for different algorithms under various lengths of trajectory segment (l) in Fig. 7. We increase the length from 5 to 30, with an equal interval of 5. As can be observed, the mean accuracy for both algorithms increases as the trajectory segment grows longer. This is probably because: with the increase of trajectory segment, more spatial context information among the GPS points can be used to improve the accuracy. This is also the key reason why offline map-matching algorithms usually have a better accuracy than online ones, since they can use the spatial information about the whole completed trajectory. What is more, SD-Matching algorithm achieves slightly higher mean accuracy under all lengths of trajectory segment than ST-Matching algorithm, demonstrating the effectiveness of bringing in the heading direction information in map-matching. We fix $k = 6$ in this experiment.

Varying k: The results for both algorithms under a different number of identified candidate edges for each GPS point (i.e., k) are shown in Fig. 8. As can be observed, the mean accuracy can be improved if we identify a bigger number of candidate mapped edges (k) for each GPS point. The reason behind is: *the probability that a GPS point can be mapped to one of candidate mapped edges correctly gets bigger with k increases, leading to a higher mean accuracy.* However, the

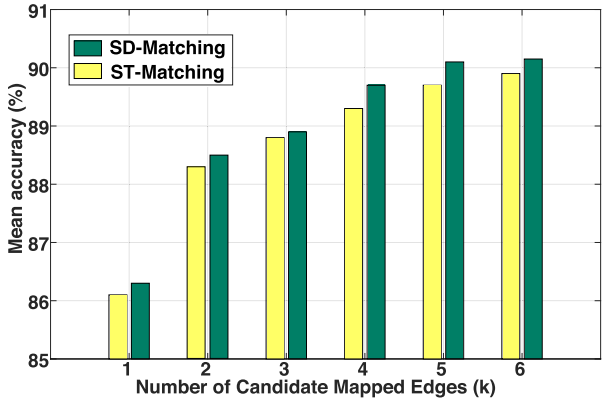


Fig. 8. Comparison results of mean accuracy for different algorithms under different ks.

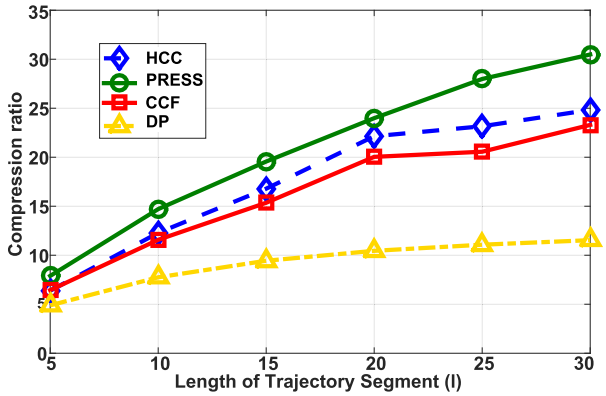


Fig. 9. Comparison results of compression ratio for different algorithms under different ls.

improvement on mean accuracy is quite limited when k increases from 4 to 6. Again, the proposed SD-Matching algorithm performs slightly better than that of ST-Matching algorithm under all ks. We fix $l = 10$ in this experiment.

In summary, comparing to ST-Matching, SD-Matching performs consistently better under different ls and ks.

2) HCC Algorithm: 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 under different ls in the following. As a comparison, the results of the baseline algorithms are also shown.

Varying l: The experimental results for different compression algorithms under different ls are presented in Fig. 9. It is observed that the compression ratio for HCC algorithm increases as the trajectory segment grows longer, which is consistent with **Theorem** 1 (see Appendix VII-B for more details). The similar tendency of compression ratio with the increase of l for the other three algorithms can be also observed.

As can be seen from Fig. 9, HCC, PRESS and CCF algorithms achieve much higher compression ratio than DP algorithm under all ls, demonstrating the superior performance of map-matching based trajectory compression algorithms. HCC algorithm achieves the compression ratio in-between

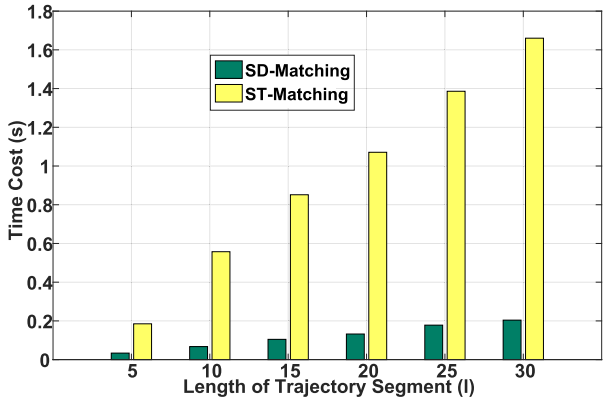


Fig. 10. Comparison results of the time cost for SD-Matching and ST-Matching under different ls.

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 [31], [56]. 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 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.

D. Efficiency Study

1) SD-Matching Algorithm: Similar to the effectiveness study, we investigate the performance on the time cost of SD-Matching under different lengths of trajectory segment (l) and different numbers of candidate mapped edges (k), respectively.

Varying l: We show the results on the time cost for SD-Matching and ST-Matching under different lengths of trajectory segment (l) in Fig. 10. More average computation time is required as the length of trajectory segment gets longer. More specifically, their time cost almost increases linearly with the length of trajectory segment, but with different slopes. The time cost for SD-Matching algorithm increases much slower than ST-Matching algorithm. In summary, SD-Matching algorithm is much more efficient and can respond within no more than 0.2 seconds for all lengths of trajectory segment. Combining results shown in Fig. 7 and Fig. 10, we can draw the conclusion that the improvement of map-matching accuracy is at the cost of increasing the computation time. When determining the value of l, we need to make a trade-off between the accuracy and the computation time. For instance,

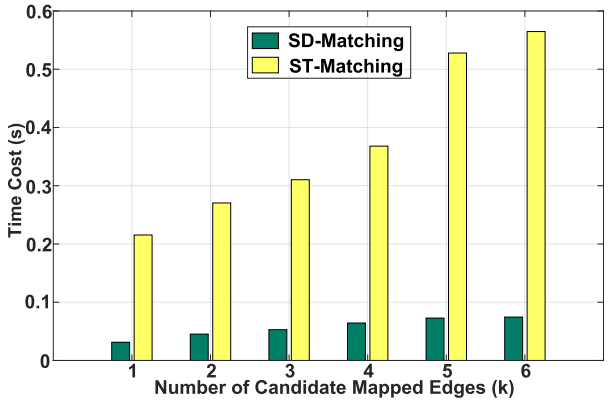


Fig. 11. Comparison results of the time cost for SD-Matching and ST-Matching under different ks.

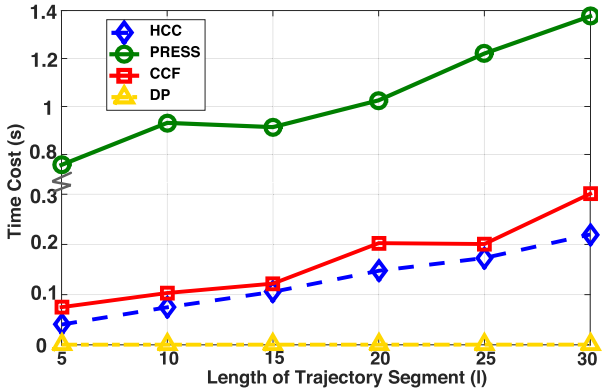


Fig. 12. Comparison results of the time cost for different trajectory compressor under different ls.

on average, for SD-Matching algorithm, a mapped trajectory can be returned within less than 0.05 seconds if we set $l = 5$, but the accuracy may be too low to be used in practical applications. We fix $k = 6$ in this experiment.

Varying k: We report the results on the time cost for SD-Matching and ST-Matching algorithms under different numbers of identified candidate mapped edges (k) for each GPS point in Fig. 11. Compared to SD-Matching, ST-Matching consumes much more time. For both algorithms, an increasing tendency can be also observed as k gets bigger. In more detail, as k gets bigger, for ST-Matching algorithm, its time cost increases exponentially, however, for SD-Matching algorithms, the time cost increases slowly. This is because: for ST-Matching algorithm, the path-finding with A* algorithm would be initialized for every pair of starting and ending nodes. The number of pairs is controlled by the value of k^2. On the contrary, for SD-Matching, the path-finding would be only initiated on top of the trees that contain both the starting and ending nodes. The number of such trees is greatly smaller than k^2 in real cases. Moreover, path-finding on top of the trees with DFS is also more efficient than A* since the searching space is much smaller, as illustrated in Fig. 5. We fix $l = 10$ in this experiment.

In a nutshell, SD-Matching performs more effectively, and also costs less time under different ls and ks simultaneously.

2) HCC Algorithm: Similar to the effectiveness study, we also test the time cost of HCC algorithm under different ls. Note that the time cost here refers to the total time cost at both phases, including trajectory map-matching and trajectory compression. In order to understand the time cost at different phases, we further show the results of detailed time distributions. We fix $k = 6$ in this experiment.

Varying l: We show the results of the time cost for HCC under different ls, as well as the results of the baseline algorithms in Fig. 12. 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.

TABLE I

THE PERCENTAGE OF TIME COST AT TWO PHASES IN HCC UNDER DIFFERENT lS

ls	5	10	15	20	25	30
Phase I (%)	83.56	89.91	89.24	89.73	89.77	**91.81**
Phase II (%)	16.44	10.90	10.76	10.27	10.23	8.19

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. 9 and Fig. 12.

Time Distribution on Different Phases: We show the time distribution of time cost on two phases (in percentage) for HCC algorithm under different ls in Table I. As can be seen from the table, most of time is consumed in the phase of map-matching, e.g., over 83% time are consumed in this phase under all lengths. Moreover, with the length of trajectory segment (l) getting longer, the time cost of Phase I takes up more. For instance, the time cost of Phase I occupies over 91% of the total time when $l = 30$. Based on the results, it is easy to get that the computation time bottleneck of the map-matching based trajectory compression is actually the map-matching. To further accelerate the online trajectory compression, a more efficient map-matching algorithm is required.

E. Boundary Choice of l

We are also interested at the boundary choice of the length of trajectory segment (i.e., l), since the performance of the proposed framework becomes better if we set a bigger value of l. 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

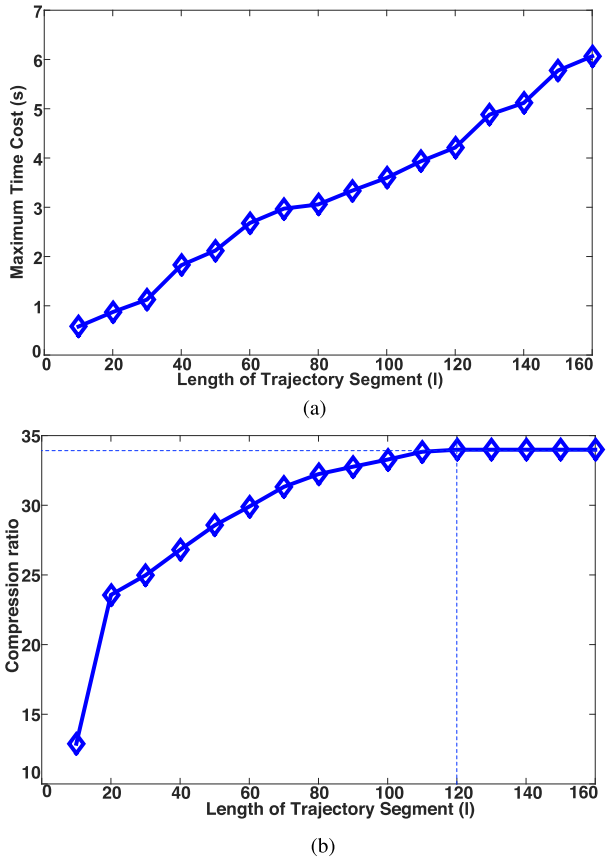


Fig. 13. Results of the boundary choice of l. (a) Maximum time costs under different ls. (b) Compression ratios under different ls.

of the GPS trajectory data (i.e., 6 seconds in our study). Thus, to get the boundary choice of l, we increase the 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 seconds.

The maximum time costs under different ls are shown in Fig. 13(a). The maximum time cost almost climbs linearly with the increase of l. The maximum time cost is close to 6 seconds when $l = 160$. We also examine the performance on the compression ratio by varying l from 10 to 160. As proofed in **Theorem** 1, the improvement room is quite limited when l increases from a large value. For instance, as shown in Fig. 13(b), 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 seconds), and the boundary choice of l is 120 in our case. We fix $k = 6$ in this experiment.

F. Other Studies

1) Varying Road Network Density: We dedicatedly divide the whole city of Beijing into two regions, i.e., region within the third ring (Region I) and region outside the third ring (Region II) respectively. The number of nodes and edges in the road network per square kilometers (i.e., density) in Region I is almost 6 times denser than that in Region II. More statistics of the two regions are further provided in Table II.

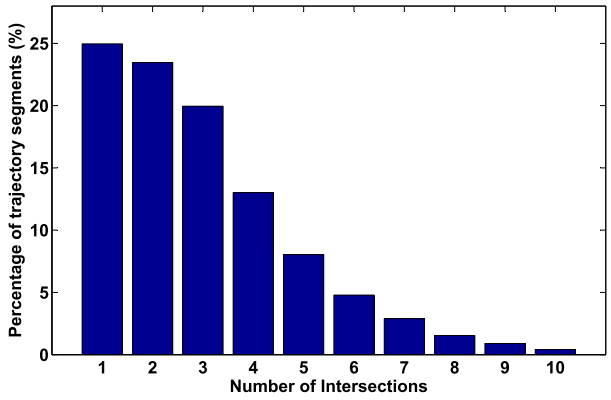
TABLE II
RESULTS UNDER DIFFERENT ROAD NETWORK DENSITIES

	Within the 3rd Ring	Outside the 3rd Ring
Area (km^2)	$\approx$ 334	$\approx$ 5992
# Nodes	38,000	103,000
# Edges	45,000	112,000
Map-matching accuracy	89.33%	92.13%
Compression Ratio	11.24	11.94
Time Cost (sec.)	0.102	0.077

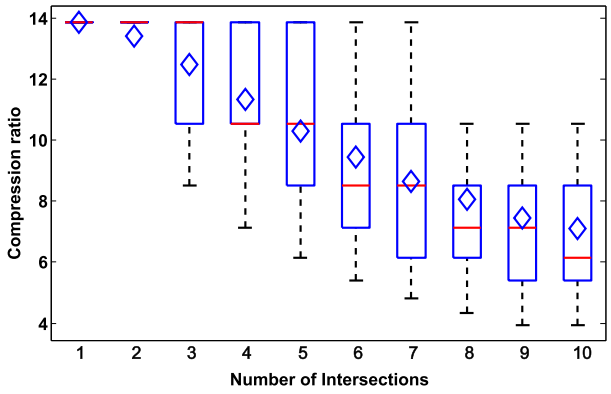
The performance results of SD-Matching and HCC algorithms are also shown, including matching accuracy, compression ratio and time cost. As can be observed, SD-Matching algorithm achieves a slightly better accuracy in Region I than Region II. Similarly, HCC algorithm achieves slightly better compression ratio but consumes less time in region with sparse road network (i.e., Region II). This is probably due to that: 1) drivers are more inclined to go straight at intersections in Region II; 2) path-finding in map-matching is much more efficient in Region II where the road network is less complicated. We fix $k = 6$, $l = 10$ in this experiment.

2) Varying Number of Intersection: We also investigate the relationship between the compression ratio and the number of intersections of the trajectory segment for HCC algorithm. For each trajectory segment, with the road network data, it is easy to know the number of intersections that it travels. The histogram of the number of intersections of all trajectory segments is shown in Fig. 14(a). The total number of trajectory segments used in this experiment is around 1,600,000. As can be seen, most of trajectory segments include less than 5 intersections. For each set of trajectory segments with the same number of intersections, we compute the corresponding compression ratio values and do some basic statistics, with the results shown in Fig. 14(b). The average value (diamond remark), in addition to the box-plot values [53] (i.e., 25th percentile, 50th percentile, 75th percentile, maximum value and minimum value) are obtained and plotted. As can be observed, the average and boxplot values of compression ratio generally decrease as the number of intersections increases. For instance, the compression ratio is quite high when the number of intersection is only one. This is because the compression ratio is tightly related with the number of intersections with remarkable heading changes, at most one out-edge is needed to be retained when the trajectory segment includes only one intersection. As the number of intersections increases, the chance that the number of intersection with remarkable heading changes getting bigger also increases, resulting in the decrease of compression ratio. As expected, the compression ratio of trajectory segments with 10 intersections is much smaller, with an average value of around 7. We fix $k = 6, l = 10$ in this experiment.

3) Varying Sampling Rate: To investigate how HCC algorithm performs under different sampling rates, we adopt a simple method based on down-sampling. The sampling rate here refers to the sampling interval of the GPS device in time. The performance results are shown in Table III.



(a)



(b)

Fig. 14. The relationship between the compression ratio and the number of intersections. (a) Distribution of intersection number of the trajectory segments. (b) Box-plot results of the compression ratio w.r.t the number of intersections.

TABLE III
RESULTS UNDER DIFFERENT SAMPLING RATES

Sampling Rate (sec.)	6	30	60	120	150
Compression Ratio	14.11	6.27	3.52	1.54	1.13
Time Cost (sec.)	0.078	0.122	0.427	0.517	0.786

The compression ratio declines significantly with the sampling interval increasing. The key reason is that the storage size of the original trajectory data decreases with the sampling time interval increases, while the size of the compressed trajectory still remains unchanged. In addition, when the sampling time interval is 150 seconds, the compression ratio is almost close to 1, implying that there is quite limited room for data compression when the data is sparse. As for the efficiency issue, more computation time is required for the trajectory data with higher sampling rates. This is probably due to the following fact: the travel distance between two consecutive GPS points gets longer with the sampling rate increases, which results in more uncertainties and complicates the map-matching consequently. We fix $k = 6, l = 10$ in this experiment.

4) 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. Readers can refer to Appendix VII-C for more details on the query location estimation and error measurement.

TABLE IV
RESULTS OF MEAN ERROR IN WHEN-AND-WHERE QUERY UNDER DIFFERENT lS

ls	5	10	15	20	25	30
mean error (m)	112.8	144.7	187.9	200.5	236.4	271.3

The results of mean error under different lengths of trajectory segment (l) are shown in Table IV. 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, making it more difficult to estimate its position at the given time. However, compared to the results reported in [23], [46], our proposed algorithm still achieves a better query quality. For instance, the mean error of our method is less than 145 meters when l equals 10, which can be usable for most of LBSs in real life. We fix $k = 6, l = 10$ in this experiment.

G. Deployment in Real Environment

We deploy our proposed framework in a Volkswagen Passat Car made in 2015. In the real implementation, we use two smartphones, one of them is used to collect GPS data, while the other is used to serve as the primary computing platform. The models of smartphones used for data collection and data computing are HUAWEI P9 and P10, respectively. 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 build-in GPS sensor to mimic the vehicle-mounted GPS devices to collect the trajectory data. 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. 15. The data stream is sent to the other smartphone for processing via the established bluetooth link. With the deployed system, we collected a trajectory dataset from 17th March to 21st April, 2018, in the city of Chongqing, China. The accumulated driving distance is around 513 kilometers. We fix $k = 6, l = 10$ and set the sampling rate to 6 seconds by default in the real implementation.

We also test the time cost at compressing each trajectory segment in the real environment, with the result shown in Fig.16. As can be observed, the system is quite time-efficient. In more detail, among all the trajectory segments, almost 80% of them can be compressed within 200 milliseconds, and the maximal time cost is around 400 milliseconds. As the sampling time interval is 6 seconds, the system can respond timely (i.e., accomplish compressing a trajectory segment before receiving the newly generated data point).

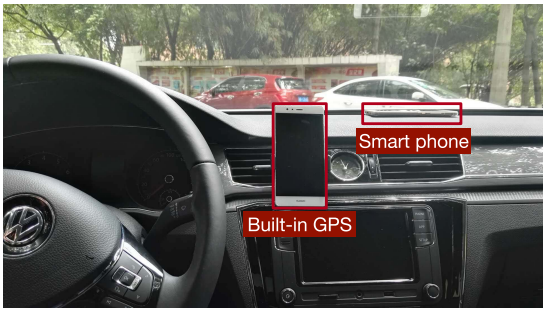


Fig. 15. Deployment of TrajCompressor system in the real environment.

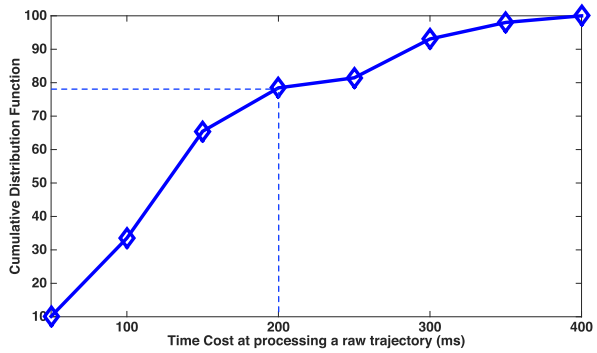


Fig. 16. CDF result of time cost for TrajCompressor system.

Therefore, the online feature of TrajCompressor can be guaranteed.

VII. Conclusion and Future Work

In this paper, we present a novel framework called TrajCompressor, with the objective of compressing trajectory data before sending it to the data center. The framework consists of two phases, i.e., SD-Matching and HCC, which take full advantage of a new dimension of information (i.e., heading direction and heading change) collected by vehicle-mounted GPS devices. Extensive results demonstrate its superior performance in the quality of map-matching and compression ratio.

In the future, we plan to broaden and deepen this work in the following directions. Since the quality of trajectory mapping is extremely essential for the map-matching based trajectory compression algorithms, our foremost future plan is to develop more advanced trajectory matching algorithms. In more detail, in our current map-matching phase, only the shortest path between two consecutive GPS points is considered, which may not correspond to the true driving path especially when the sampling time interval is big. Hence, the driving preference for each individual driver should be integrated when inferring the path in-between. Moreover, it is well-recognized that the currently available road network is often inaccurate and incomplete, thus, how to develop new map-matching algorithms that can overcome errors caused by the road network is also worth to be explored. Secondly, we plan to improve the compression performance further by taking more actions, including compression enhancement, more advanced edge encoding methods. Thirdly, we also intend to investigate the trajectory compression in the temporal dimension. For instance, the accelerating (decelerating) behaviors of drivers may be retained. More specifically, the mean accelerated speed (a) of adjacent two consecutive GPS points can be easily calculated. Then, we can get a series of two-dimensional data (t_i, a_i), 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 t_i. Afterwards, we could utilize some common methods to compress the two-dimensional data, such as Huffman coding. Last but not least, to enable more and smarter urban services, we intend to explore the potentials of the combination research of trajectory compression and trajectory summarization [1], [14], [42], [47], [59], since trajectory summarization is capable of making the trajectory data more intuitively and more understandable.

TABLE V
Commonly Used Acronyms in the Paper

Acronym	Full Form
TrajCompressor	Trajectory Compressor
GPS	Global Positioning System
SD-Matching	Spatio-Directional Matching
HCC	Heading Change Compression
ST-Matching	Spatio-Temporal Matching
PRESS	Paralleled Road-Network-Based Trajectory Compression
CCF	Clockwise Compression Framewore
DP	Douglas-Peucker
FEC	Frequent Edge Compression

Appendix

A. Acronyms and Full Names

Table V summarizes the commonly used acronyms and the corresponding full names in the paper.

B. Theorem

Theorem 1: The compression ratio (cr) of HCC increases with the length of trajectory segment. However, the increment becomes marginal when the length increases from a bigger value.

Proof: We suppose that the raw trajectory (T) contains L GPS points, which is split to n raw trajectory segments with an equal length of l, i.e., $N = nl$. Each raw trajectory segment can be denoted by $\tau_i = \langle p_i, \cdots, p_{i+l-1} \rangle$. τ_{ci} represents its corresponding compressed trajectory segment. Thus, the compression ratio can be determined by the following Eq. 9.

$$cr = \frac{space(T)}{\sum_{i=1}^{n} space(\tau_{ci})} \tag{9}$$

where $space(T)$ refers to the storage space that occupied by the raw trajectory; $space(\tau_{ci})$ refers to the storage space that occupied by the compressed trajectory segment τ_{ci}. With HCC, the space of each compressed trajectory is the total space of all the edges in each compressed trajectory. As can be observed in Algorithm 1, the starting and ending edges are always maintained for each trajectory segment. Thus, $space(\tau_{ci})$ can be represented by Eq. 10.

$$space(\tau_{ci}) = (2 + x_i)\ space(edge) \tag{10}$$

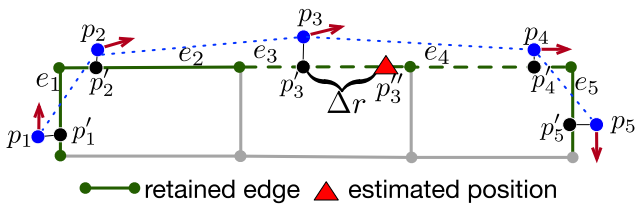


Fig. 17. Illustrative example of location estimation and error measurement in *when-and-where* query.

where $space(edge)$ refers to the space occupied by one edge, which can be viewed as constant and determined by the encoding manner. In this study, each edge is simply encoded by a unique 9 digits. $2 + x_i$ refers to the number of edges in the compressed trajectory τ_{ci}.

By putting Eq. 10 into Eq. 9, we can get:

$$cr = \frac{space(T)}{2n\ space(edge) + \sum_{i=1}^{n} x_i\ space(edge)} \tag{11}$$

For the raw trajectory T, the term $\sum_{i=1}^{n} x_i\ space(edge)$ would keep fixed which is determined by the total number of retained edges (excluding starting and ending edges), regardless of the length of split trajectory segment (i.e., l). The number is just the number of out-edges with remarkable heading changes. We denote the fixed space as C. When the length of trajectory segment increases from l to $l + \Delta l$, the difference between the compression ratio can be represented by the following Eq. 12.

$$\frac{cr_{i+1}}{cr_i} = \frac{2n\ space(edge) + C}{2n'\ space(edge) + C} \tag{12}$$

where n' is the number of trajectory segments if we set the length of trajectory segment to $l + \Delta l$. It is easy to get that $n' < n$, thus $\frac{cr_{i+1}}{cr_i} > 1$ always holds, *indicating the compression ratio increase with* l.

What is more, both n and n' are quite small if l is big, thus C will dominate both numerator and denominator in Eq. 12. In this case, we have $\frac{cr_{i+1}}{cr_i} \to 1$, *demonstrating the improvement of compression ratio is quite limited when* l *increases from a big value*. ■

C. Location Estimation and Error Measurement

As defined in **Definition** 5, only the time when the vehicle enters the first edge of the compressed trajectory is recorded (i.e., the entering time exactly equals the sampling time of the first GPS point in the raw trajectory segment). With the time label of the compressed trajectory, it is trivial to find the corresponding one where the interested vehicle locates at the user-specified time.

We suppose that $\{t_1, \langle e_1, e_2, e_5\rangle\}$ is the corresponding compressed trajectory for the user-specified time t_3 in the illustrative case shown in Fig. 17. It is easy to recover its original trajectory (i.e., $\langle e_1, e_2, e_3, e_4, e_5\rangle$). To estimate the location of the vehicle at the user-specified time t_3, we assume that the vehicle moves at a constant speed from the starting edge to the ending edge in the original trajectory. The travel distance of the vehicle is the total length of the original trajectory. The travel time of the vehicle when driving through the compressed trajectory is just the time difference between its time and the time label of its successor. Thus, the constant speed can be also easily obtained. With the time difference (i.e., $t_3 - t_1$) and the obtained travel speed, we can estimate the vehicle's location (e.g., p_3'' in the example).

The true location of the vehicle is just the projected one in the road network (e.g., p_3' is the true location of p_3). The error is measured by the distance (Δr) between the estimated location and the true location. Similarly, the mean error which is the average value of errors of all user queries is used to evaluate the overall quality of response for the user-query.

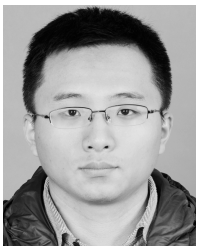
REFERENCES

[1] S. Andrae and S. Winter, "Summarizing GPS trajectories by salient patterns," Tech. Rep., 2005.
[2] J. L. Bentley and H. A. Maurer, "Efficient worst-case data structures for range searching," *Acta Inf.*, vol. 13, no. 2, pp. 155–168, 1980.
[3] D. Bernstein and A. Kornhauser, "An introduction to map matching for personal navigation assistants," Tech. Rep., 1998.
[4] M. Bierlaire, J. Chen, and J. Newman, "A probabilistic map matching method for smartphone GPS data," *Transp. Res. C, Emerg. Technol.*, vol. 26, pp. 78–98, Jan. 2013.
[5] S. Brakatsoulas, D. Pfoser, R. Salas, and C. Wenk, "On map-matching vehicle tracking data," in *Proc. Int. Conf. Very Large Databases*, 2005, pp. 853–864.
[6] P. S. Castro, D. Zhang, and S. Li, "Urban traffic modelling and prediction using large scale taxi GPS traces," in *Proc. Int. Conf. Pervasive Comput.*, 2012, pp. 57–72.
[7] C. Chen *et al.*, "MA-SSR: A memetic algorithm for skyline scenic routes planning leveraging heterogeneous user-generated digital footprints," *IEEE Trans. Veh. Technol.*, vol. 66, no. 7, pp. 5723–5736, Jul. 2017.
[8] 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.
[9] C. Chen *et al.*, "iBOAT: Isolation-based online anomalous trajectory detection," *IEEE Trans. Intell. Transp. Syst.*, vol. 14, no. 2, pp. 806–818, Jun. 2013.
[10] 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.
[11] Y. Chen, K. Jiang, Y. Zheng, C. Li, and N. Yu, "Trajectory simplification method for location-based social networking services," in *Proc. Int. Workshop Location Based Social Netw.*, 2009, pp. 33–40.
[12] Y. Ding, C. Chen, S. Zhang, B. Guo, Z. Yu, and Y. Wang, "GreenPlanner: Planning personalized fuel-efficient driving routes using multi-sourced urban data," in *Proc. IEEE Int. Conf. Pervasive Comput. Commun. (PerCom)*, Mar. 2017, pp. 207–216.
[13] 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. Geograph. Inf. Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.
[14] L. Etienne, T. Devogele, M. Buchin, and G. McArdle, "Trajectory box plot: A new pattern to summarize movements," *Int. J. Geograph. Inf. Sci.*, vol. 30, no. 5, pp. 835–853, 2016.
[15] Z. Feng and Y. Zhu, "A survey on trajectory data mining: Techniques and applications," *IEEE Access*, vol. 4, pp. 2056–2067, 2016.
[16] C. Y. Goh, J. Dauwels, N. Mitrovic, M. T. Asif, A. Oran, and P. Jaillet, "Online map-matching based on hidden Markov model for real-time traffic sensing applications," in *Proc. IEEE Conf. Intell. Transp. Syst. (ITSC)*, Sep. 2012, pp. 776–781.
[17] J. S. Greenfeld, "Matching GPS observations to locations on a digital map," in *Proc. 81st Annu. Meeting Transp. Res. Board*, 2002, vol. 1, no. 3, pp. 164–173.
[18] Y. Gu, Z. S. Qian, and F. Chen, "From Twitter to detector: Real-time traffic incident detection using social media data," *Transp. Res. C, Emerg. Technol.*, vol. 67, pp. 321–342, Jun. 2016.
[19] J. Gudmundsson, J. Katajainen, D. Merrick, C. Ong, and T. Wolle, "Compressing spatio-temporal trajectories," *Comput. Geometry*, vol. 42, no. 9, pp. 825–841, 2009.

[20] W. Han, Z. Deng, J. Chu, J. Zhu, P. Gao, and T. Shah, "A parallel online trajectory compression approach for supporting big data workflow," *Computing*, vol. 100, pp. 3–20, Jan. 2017.
[21] Y. Han, W. Sun, and B. Zheng, "COMPRESS: A comprehensive framework of trajectory compression in road networks," *ACM Trans. Database Syst. (TODS)*, vol. 42, no. 2, 2017, Art. no. 11.
[22] 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. (ICPADS)*, Dec. 2016, pp. 256–263.
[23] 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*, Dec. 2016, pp. 472–481.
[24] W. Kang, S. Li, W. Chen, K. Lei, and T. Wang, "Online map-matching algorithm using object motion laws," in *Proc. IEEE Int. Conf. High Perform. Smart Comput. (HPSC)*, May 2017, pp. 249–254.
[25] G. Kellaris, N. Pelekis, and Y. Theodoridis, "Trajectory compression under network constraints," in *Advances in Spatial and Temporal Databases*. Berlin, Germany: Springer, 2009, pp. 392–398.
[26] G. Kellaris, N. Pelekis, and Y. Theodoridis, "Map-matched trajectory compression," *J. Syst. Softw.*, vol. 86, no. 6, pp. 1566–1579, 2013.
[27] E. Keogh, S. Chu, D. Hart, and M. Pazzani, "An online algorithm for segmenting time series," in *Proc. IEEE Conf. Data Mining*, Nov./Dec. 2001, pp. 289–296.
[28] H. Koller, P. Widhalm, M. Dragaschnig, and A. Graser, "Fast hidden Markov model map-matching for sparse and noisy trajectories," in *Proc. IEEE Conf. Intell. Transp. Syst. (ITSC)*, Sep. 2015, pp. 2557–2561.
[29] M. Kubička, A. Cela, H. Mounier, and S.-I. Niculescu, "Comparative study and application-oriented classification of vehicular map-matching methods," *IEEE Intell. Transp. Syst. Mag.*, vol. 10, no. 2, pp. 150–166, Apr. 2018.
[30] W.-H. Lee, S.-S. Tseng, J.-L. Shieh, and H.-H. Chen, "Discovering traffic bottlenecks in an urban network by spatiotemporal data mining on location-based services," *IEEE Trans. Intell. Transp. Syst.*, vol. 12, no. 4, pp. 1047–1056, Dec. 2011.
[31] B. Li *et al.*, "Hunting or waiting? Discovering passenger-finding strategies from a large-scale real-world taxi dataset," in *Proc. IEEE Conf. Pervasive Comput. Commun. Workshops (PerCom Workshops)*, 2011, pp. 63–68.
[32] X. Lin, S. Ma, H. Zhang, T. Wo, and J. Huai, "One-pass error bounded trajectory simplification," *Proc. VLDB Endowment*, vol. 10, no. 7, pp. 841–852, 2017.
[33] 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. (ICDE)*, Apr. 2015, pp. 987–998.
[34] C. Long, R. C.-W. Wong, and H. V. Jagadish, "Trajectory simplification: On minimizing the direction-based error," *Proc. VLDB Endowment*, vol. 8, no. 1, pp. 49–60, 2014.
[35] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang, "Map-matching for low-sampling-rate GPS trajectories," in *Proc. ACM SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, 2009, pp. 352–361.
[36] C. Lv, F. Chen, Y. Xu, J. Song, and P. Lv, "A trajectory compression algorithm based on non-uniform quantization," in *Proc. Int. Conf. Fuzzy Syst. Knowl. Discovery (FSKD)*, 2015, pp. 2469–2474.
[37] N. Meratnia and R. A. de By, "Spatiotemporal compression techniques for moving point objects," in *Proc. Int. Conf. Extending Database Technol.*, 2004, pp. 765–782.
[38] J. Muckell, P. W. Olsen, Jr., J.-H. Hwang, C. T. Lawson, and S. Ravi, "Compression of trajectory data: A comprehensive evaluation and new approach," *GeoInformatica*, vol. 18, no. 3, pp. 435–460, 2014.
[39] P. Newson and J. Krumm, "Hidden Markov map matching through noise and sparseness," in *Proc. Int. Conf. Adv. Geograph. Inf. Syst.*, 2009, pp. 336–343.
[40] A. Nibali and Z. He, "Trajic: An effective compression system for trajectory data," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 11, pp. 3138–3151, Nov. 2015.
[41] O. Pink and B. Hummel, "A statistical approach to map matching using road network geometry, topology and vehicular motion constraints," in *Proc. IEEE Conf. Intell. Transp. Syst.*, Oct. 2008, pp. 862–867.
[42] Q. Qu, S. Liu, F. Zhu, and C. S. Jensen, "Efficient online summarization of large-scale dynamic networks," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 12, pp. 3231–3245, Dec. 2016.
[43] M. A. Quddus, W. Y. Ochieng, and R. B. Noland, "Current map-matching algorithms for transport applications: State-of-the art and future research directions," *Transp. Res. C, Emerg. Technol.*, vol. 15, no. 5, pp. 312–328, 2007.
[44] K.-F. Richter, F. Schmid, and P. Laube, "Semantic trajectory compression: Representing urban movement in a nutshell," *J. Spatial Inf. Sci.*, vol. 2012, no. 4, pp. 3–30, 2012.
[45] A. Silva, R. Raghavendra, M. Srivatsa, and A. K. Singh. (2016). "Prediction-based online trajectory compression." [Online]. Available: https://arxiv.org/abs/1601.06316
[46] R. Song, W. Sun, B. Zheng, and Y. Zheng, "PRESS: A novel framework of trajectory compression in road networks," in *Proc. VLDB Endowment*, vol. 7, no. 9, pp. 661–672, 2014.
[47] H. Su *et al.*, "Making sense of trajectory data: A partition-and-summarization approach," in *Proc. IEEE 31st Int. Conf. Data Eng. (ICDE)*, Apr. 2015, pp. 963–974.
[48] P. Sui and X. Yang, "A privacy-preserving compression storage method for large trajectory data in road network," *J. Grid Comput.*, vol. 16, no. 2, pp. 229–245, Feb. 2018.
[49] P. Sun, S. Xia, G. Yuan, and D. Li, "An overview of moving object trajectory compression algorithms," *Math. Problems Eng.*, vol. 2016, Apr. 2016, Art. no. 6587309.
[50] R. Tarjan, "Depth-first search and linear graph algorithms," in *Proc. Symp. Switching Automata Theory*, 1971, pp. 114–121.
[51] H. Wei, Y. Wang, G. Forman, Y. Zhu, and H. Guan, "Fast Viterbi map matching with tunable weight functions," in *Proc. ACM SIGSPATIAL Int. Conf. Adv. Geograph. Inf. Syst.*, 2012, pp. 613–616.
[52] C. E. White, D. Bernstein, and A. L. Kornhauser, "Some map matching algorithms for personal navigation assistants," *Transp. Res. C, Emerg. Technol.*, vol. 8, nos. 1–6, pp. 91–108, 2000.
[53] D. F. Williamson, R. A. Parker, and J. S. Kendrick, "The box plot: A simple visual method to interpret data," *Ann. Internal Med.*, vol. 110, no. 11, pp. 916–921, 1989.
[54] J. Yuan, Y. Zheng, C. Zhang, X. Xie, and G.-Z. Sun, "An interactive-voting based map matching algorithm," in *Proc. IEEE Conf. Mobile Data Manage. (MDM)*, May 2010, pp. 43–52.
[55] D. Zhang, N. Li, Z.-H. Zhou, C. Chen, L. Sun, and S. Li, "iBAT: Detecting anomalous taxi trajectories from GPS traces," in *Proc. Int. Conf. Ubiquitous Comput.*, 2011, pp. 99–108.
[56] 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. 2014.
[57] K. Zheng, Y. Zheng, X. Xie, and X. Zhou, "Reducing uncertainty of low-sampling-rate trajectories," in *Proc. IEEE Int. Conf. Data Eng. (ICDE)*, Apr. 2012, pp. 1144–1155.
[58] Y. Zheng, "Trajectory data mining: An overview," *ACM Trans. Intell. Syst. Technol.*, vol. 6, no. 3, p. 29, 2015.
[59] F. Zhou, Q. Qu, and H. Toivonen, "Summarisation of weighted networks," *J. Exp. Theor. Artif. Intell.*, vol. 29, no. 5, pp. 1023–1052, 2017.
[60] X. Zhou, Y. Ding, H. Tan, Q. Luo, and L. M. Ni, "HIMM: An HMM-based interactive map-matching system," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2017, pp. 3–18.

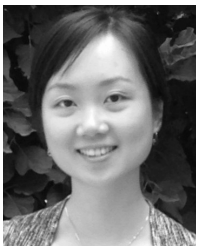
Chao Chen received the B.Sc. and M.Sc. degrees in control science and control engineering from Northwestern Polytechnical University, Xi'an, China, in 2007 and 2010, respectively, and the Ph.D. degree from Pierre and Marie Curie University and the Institut Mines-Télécom/Télécom SudParis, France, in 2014.

In 2009, he was a Research Assistant with The Hong Kong Polytechnic University, Kowloon, Hong Kong. He is currently a Full Professor with the College of Computer Science, Chongqing University, Chongqing, China. He has published over 80 papers, including 20 ACM/IEEE TRANSACTIONS. His research interests include pervasive computing, mobile computing, urban logistics, data mining from large-scale GPS trajectory data, and big data analytics for smart cities. His work on taxi trajectory data mining was featured by the *IEEE Spectrum* in 2011 and 2016. He was a recipient of the Best Paper Runner-Up Award at the MobiQuitous 2011.



Yan Ding received the B.Sc. degree from the College of Mechanical Engineering, Chongqing University, China, in 2016, where he is currently pursuing the master's degree with the College of Computer Science. His research interests include route planning, map matching, and spatial-temporal trajectory compression.

Xuefeng Xie received the B.A. degree in film art from the Sichuan Fine Art Institute, Chongqing, China, in 2012, and M.A. degree (Hons.) from the School of Media and Communication, University of Leeds, Leeds, U.K. She is currently a Research Assistant with the College of Computer Science, Chongqing University, China. Her research interests include data visualization, aesthetic beauty, and quantitative analysis.



Shu Zhang received the bachelor's degree from the Civil Aviation University of China, Tianjin, China, in 2007, the master's degree from Mississippi State University, USA, in 2010, and the Ph.D. degree in management sciences from the University of Iowa, USA, in 2015. She is currently an Assistant Professor with the College of Economics and Business Administration, Chongqing University, Chongqing. Her research interests include vehicle routing, urban logistics, and transportation network design.

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 Northwestern Polytechnical University. His research interests include pervasive computing, mobile social network analysis, and mobile healthcare.

Liang Feng received the Ph.D. degree from the School of Computer Engineering, Nanyang Technological University, Singapore, in 2014. He was a Post-Doctoral Research Fellow with the Computational Intelligence Graduate Lab, Nanyang Technological University. He is currently an Assistant Professor with the College of Computer Science, Chongqing University, China. His research interests include computational and artificial intelligence, memetic computing, big data optimization and learning, as well as transfer learning.