Clockwise Compression for Trajectory Data under Road Network Constraints

Yudian Ji †, Yuda Zang ‡, Wuman Luo #, Xibo Zhou †, Ye Ding †, Lionel M. Ni#
†*Hong Kong University of Science and Technology*
‡*Tsinghua University*
#*University of Macau*
{*yjiab, xzhouaa, valency*}*@ust.hk, zangyd13@mails.tsinghua.edu.cn,* {*wumanluo, ni*}*@umac.mo*

***Abstract*—Big trajectory data introduces severe challenges for data storage and communication. In this paper, we propose a novel compression framework called *Clockwise Compression Framework* (CCF) for big trajectory data compression under road network constraints. In CCF, we design several new methods: 1) a spatial compression algorithm called Enhanced Clockwise Encoding (ECE), 2) a temporal compression algorithm called Fitting-based Temporal Simplification (FTS), and 3) a dedicated querier that processes queries based on the above spatial and temporal compression algorithms, without fully decompressing the trajectroy data. By leveraging the topological information of the road network, CCF is able to perform both spatial compression and temporal compression in on-line modes. We perform extensive experiments in a real big trajectory dataset to verify both effectiveness and efficiency of our methods. CCF shows promising performances in various metrics and outperforms the state-of-the-art methods.**

I. Introduction

The popularization of location-aware devices has generated large amounts of trajectory data. Typically, a spatio-temporal trajectory consists of a series of location points and timestamps, indicating the curve that a moving object travels. Trajectory data is crucial in many real-world applications, such as traffic control [1], driving behavior analysis [2], motion prediction [3], etc. However, their ever-growing volume causes the crisis of data storage and communication. For example, the trajectory data collected from 59700 taxis in Beijing within only 3 months has the size around 1TB. Obviously, we need an effective trajectory compression method that can largely reduce the size of a trajectory and keep its utility. Ideally, a desirable trajectory compression method have at least three qualities: good compression ratio, high compression efficiency, and low query overhead. In this paper, we focus on big trajectory data compression under road network constraints.

Nowadays, trajectory data is often aligned to the road network, which increases location accuracy and reduces data redundancy [4] [5]. The spatial trajectory is therefore mapped to a path in a road network. This preprocessing of representation increases the difficulty of trajectory compression. Besides, on-line processing is becoming increasingly essential in various modern applications, such as urban sensing [6], traffic analyzing [1], etc. This calls for the capabilities of on-line data compression and efficient data access.

Unfortunately, existing methods cannot handle such problems effectively. So far, only a few methods have been proposed for trajectory compression under road network constraints [7] [8] [9] [10]. Most of them show limited performances in compression ratio, compression efficiency and query overhead. The state-of-the-art off-line compression framework, namely PRESS [10], achieves better overall performance. However, it contains off-line computations in its major steps, such as Huffman coding, which greatly affects its on-line compression capacity. Although such algorithms can be adapted to on-line version by using methods like adaptive variation or directly using the pre-trained Huffman tree, its performance drops abruptly. Moreover, some auxiliary structures adopted in previous methods are gigantic for complex road networks. The last-edge shortest path table adopted in PRESS has the size of 110.6GB in terms of the road network of Beijing.

In this paper, we present ***C**lockwise **C**ompression **F**ramework* (CCF), a novel framework for trajectory compression under road network constrains. Compared with PRESS, CCF achieves a better compression performance. Moreover, CCF can be easily adapted to compressing on-line data streams. In CCF, we propose a topology-based lossless compression algorithm called *Enhanced Clockwise Encoding* (ECE) for spatial compression, and an error-bounded line simplification algorithm called *Fitting-based Time Simplification* (FTS) for temporal compression. We study the performance of CCF extensively on a real big trajectory dataset. Overall, we have made the following contributions:

- We propose a compression framework, namely Clockwise Compression Framework (CCF), for map-matched trajectory compression under road network constraints. It contains a lossless spatial compression method and an error-bounded temporal compression method. CCF has strong overall compression performance and on-line capability.
- We propose a novel dedicated encoding algorithm for spatial information of the trajectory, namely Clockwise Encoding, which creatively leverages the topological information of road networks. CE compresses the trajectory without acquiring the preknowledge of the input

978-1-4673-9005-7/16/$31.00 ©2016 IEEE

data, which contributes to the efficiency and on-line capability of the framework.

- We propose three enhancement algorithms, namely Straight Path Enhancement, Frequent Follower Enhancement and Hotspot Frequent Compressive Path Enhancement to boost the compression power of Clockwise Encoding. We develop a method of mining frequent paths dedicated for compression. By using the FCP function in this paper, both frequencies and lengths of the original paths will be taken into consideration and a relation between the attributes of paths and the actual compression performance is built.
- We test our methods through real-world trajectory data. CCF achieves in average 25% higher compression ratio, 4.61 times faster compression efficiency and 43% query speedup compared with the state-of-the-art methods [10]. We also conduct an experiment in on-line scheme to advocate the on-line capability of CCF.

The rest of the paper is organized as follows: Related work of our research and important definitions will be given in Section II and Section III respectively. Section IV introduces the detailed algorithms of CCF. We provide a thorough experiment study and performance comparison in Section V. In Section VI, we give a conclusion of the paper.

II. Related Work

Trajectory compression methods can be classified into two categories based on the two kinds of problems they solve, namely line simplification and map-matched compression. Line simplification methods focus on compressing raw trajectory data represented by a sequence of GPS points, while map-matched trajectory compression methods focus on compressing the map-matched paths of trajectory data.

A. Line Simplification

Line simplification methods [11] [12] aim at using the minimum number of points to approximate the whole curve with least information loss. They usually keep an error bound to decide whether a point can be compressed or not. For trajectory data, line simplification methods leverage the geometry characteristics to reserve the spatial information. Line simplification methods are commonly used in many applications like moving pattern study [13] and motion prediction [3] where the shape of the curve is stressed.

B. Map-matched Trajectory Compression

Map-matched trajectory compression tries to find the redundancy within trajectories by matching the trajectory points to the digital map and then deal with the map-matched data. With the extra information provided by digital maps, some redundancies and patterns that cannot be derived before will be discovered after map-matching. Moreover, the trajectories are represented by paths rather than location points. The above two facts allow many new methods to be used when compressing map-matched trajectories.

Nonmaterial [9] tries to relate raw timestamps with points on the road network. It will then use less timestamps on the network to estimate the original ones with an error bound. This introduces some information loss but compresses the time information of map-matched trajectory. Routing Algorithm [7] encodes map-matched trajectory data in two ways. First, it skips a subset of consecutive road segments if such subset is the shortest path from its beginning to its end. Second, it skips a subset of consecutive road segments if each pair consecutive road segments within the subset follows the rule of least deviation. Map-matched Trajectory Compression (MMTC) [8] replaces part of the map-matched trajectory with paths with less intersections. PRESS [10] applies shortest path compression [7] as their first step. Then, it uses the global map-matched trajectory data to form a prefix tree. It adpots a prefix tree to mine all the frequences of the sub-trajectories with length less than or equal to 3. At last, it treats each sub-trajectory as a node to conduct Huffman Coding. PRESS also modifies a line simplification algorithm to compress the timestamps.

III. Definitions

To make things clear, we state the important definitions in this paper.

Definition 1. Road Segment: A road segment e_n is a directed edge between two intersections v_i and v_j. There is no other intersection on e_n.

Definition 2. Road Network: A road network RN is a directed graph $G(V, E)$, where V is the set of intersections and E is the set of road segments.

Definition 3. Path: A path $P = \{e_1, e_2, e_3.....e_n\}$ is a sequence of consequtive road segments.

Definition 4. Map-matched Trajectory: Given a trajectory T, a map-matched trajectory $T_{matched}$ is represented as $(P_{matched}, \mathfrak{t})$, where $P_{matched}$ is the map-matched path of T, and $\mathfrak{t} = (t, d)$ are the timestamps of T. d is the travelled distance ($d_{traveled}$) from the starting point (p_{start}) of the path to the perpendicular projection ($p_{project}$) onto the path of the original location point p.

IV. Methodology

In this section, we go into detail with our compression methods. We first provide an overview of Clockwise Compression Framework, then we introduce the specific algorithms corresponding to each component.

A. CCF Overview

As shown in Figure 1, CCF consists of three components, namely preprocessor, compression engine and query executor. Preprocessor applies a map-matching algorithm that

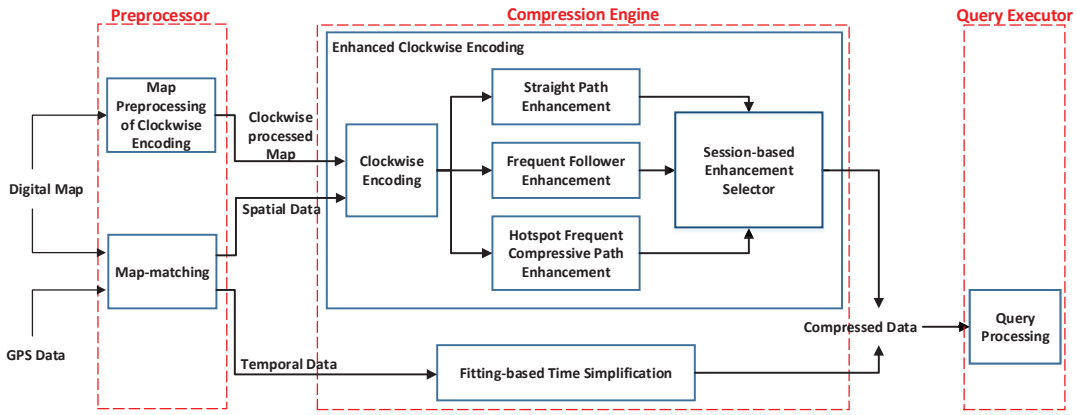

Figure 1: Overview of CCF

aligns raw trajectories with the road network, and also the map-preprocessing stage prior to a spatial compression algorithm, namely Clockwise Encoding. Compression engine compresses the map-matched trajectory, where the spatial and temporal information will be processed separately. After compression, a query executor is provided to the users to conduct queries on the compressed data.

CCF employs individual compression algorithms for spatial and temporal dimensions of trajectory data. For temporal compression, CCF uses a stand-alone algorithm named *Fitting-based Temporal Simplification* (FTS) . For spatial compression, CCF utilizes a hybrid algorithm named *Enhanced Clockwise Encoding* (ECE). As a hybrid compression algorithm, ECE is constructed by an encoding algorithm called *Clockwise Encoding* (CE); three enhancement techniques called *Straight Path Enhancement* (SPE), *Frequent Follower Enhancement* (FFE) and *Frequent Compressive Path Enhancement* (FCPE); and an ensemble strategy called *Session-based Enhancement Selector*. The algorithms implemented in CCF will be introduced in Section IV-B and IV-C.

B. Spatial Compression

This section introduces the algorithms for spatial compression.

1) Clockwise Encoding: CE compresses the map-matched path by leveraging the topological information of the road network. In real-life road networks, the number of out-going edges is limited for each intersection, and it is usually small. We have conducted an experiment using the digital map of Beijing. As shown in Table I, only 3 out of 166304 intersections are connected to more than 6 out-going road segments. All of the intersections have out-going road segments less than 8. CE is designed from the above observation. It takes an once-and-for-all preprocessing stage on the digital map before the input is encoded. For each intersection v_n in the digital map, we encode every road segment e_n that goes out of the intersection by clockwise order, as shown in Figure 2. Since the number of out-going road segments from one intersection is limited, we

Table I: Number of Out-going Road Segments From One Intersection

No. of Road Segments	<7	7	8	Total
No. of Intersections	166301	2	1	166304

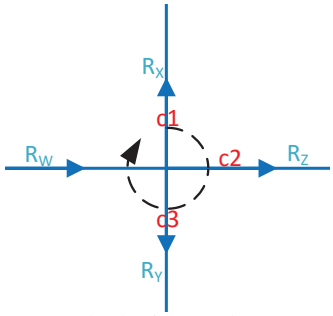

Figure 2: Clockwise Code Generation

use a clockwise code c_n, which is a binary symbol of fixed-length bit string, to represent each road segment. For each intersection, each out-going road segment is uniquely represented by a clockwise code. Note that a same clockwise code could represent different road segments for different intersections. For those special intersections with the number of out-going road segments that exceeds the limit, we judge the intersection as two intersections with a virtual link of distance 0. Thus, the two intersections will have decreased number of out-going road segments. This step can be repeated until the requirement is met.

During compression, CE first outputs the first road segment e_{start} of the trajectory in the uncompressed form. The first road segment indicates where the path starts, thus cannot be clockwise encoded. After e_{start}, the algorithm finds the endpoint of it and searches in the clockwise code table corresponding to the intersection ID of this endpoint, where the clockwise code of the next road segment will be found. The algorithm then copies the corresponding clockwise code c_n from the table to the output. This process is repeated until the whole path is clockwise encoded. CE yields a complexity of $O(|e|)$, where $|e|$ is the length of

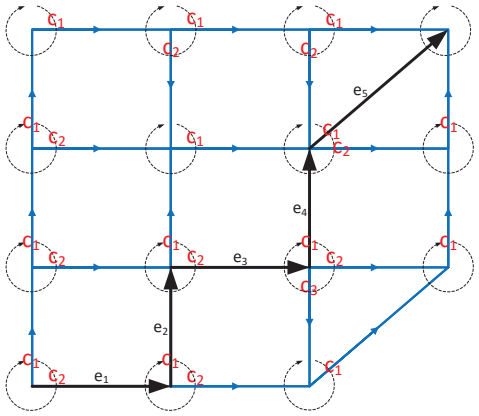

Figure 3: An Example of Clockwise Encoding

the map-matched path. An example is shown in Figure 3. The path $\{e_1, e_2, e_3, e_4, e_5\}$ can be compressed by CE as $\{e_1, c_1, c_2, c_1, c_1\}$. Since the ID of e_n is a global ID that is unique among all the road segments, it is extremely large compared with c_n. Thus, the path can be compressed into a form with less size. CE reserves the information of sequential road segments one by one, which opens the gate to further enhancements based on sub-paths.

2) Straight Path Enhancement: SPE is a variant of the follower encoding [7]. It is based on the assumption that drivers tend to follow a path with least direction deviation when driving, since it is faster to take the straight way to the destination. When a moving object travels in RN, it always goes from a in-coming road segment e_i to an out-going road segment e_j through an intersection v_n. Thus, there is an angular change $\Delta\theta$ from the in-coming road segment to the out-going road segment. With different e_j chosen, $\Delta\theta$ will be different. The idea of SPE is to compress the path with least $\Delta\theta$ at each v_n it travels through. Such path will be skipped directly since we can easily derive the path from both ends.

In our case, it is difficult to locate the end-point of a sub-path, since each road segment is derived from its former ones. Instead of skipping a straight path directly, we reserve a tuple $< s_{SPE}, \delta >$. s_{SPE} is a special binary code that indicates SPE is invoked, while δ is an offset indicating the number of road segments compressed. We also maintain an SPE table denoting a road segment and its least deviation follower to boost the compression efficiency, since such information can be retrieved once-and-for-all from the road network only. Some encoding space of clockwise codes is reserved for the special codes of the enhancements, so there will be no difficulty distinguishing special codes from the clockwise codes when decoding. For example, a path $\{e_6, e_7, e_8, e_9, e_{10}, e_{11}, e_{12}\}$ contains a sub-path $\{e_7, e_8, e_9, e_{10}\}$ with least deviation change. It is compressed to $\{e_6, c_7, c_8, c_9, c_{10}, c_{11}, c_{12}\}$ by CE. It can be further enhanced to $\{e_6, c_7, < s_{SPE}, \delta >, c_{11}, c_{12}\}$ by SPE. The algorithm runs in $O(|e|)$, where $|e|$ is the length of the input path.

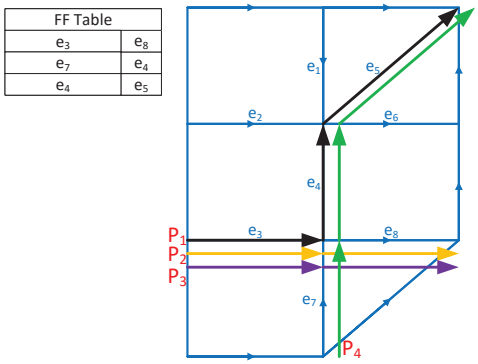

Figure 4: Frequent Follower Table Generation

3) Frequent Follower Enhancement: Frequent Follower Enchancement (FFE) is a variant of MFFC [14], which addresses the relationship between a road segment and its following road segments. The basic idea of FFE is to find the most frequent following road segment for each road segment, and utilize it for compression.

FFE first scans through the sampled data to find the most frequent following road segment for each involved road segment. The information is stored in a frequent follower (FF) table. As shown in Figure 4, there are three paths P_1=$\{e_3, e_4, e_5\}$, P_2=$\{e_3, e_8\}$, P_3=$\{e_3, e_8\}$, P_4=$\{e_7, e_4, e_5\}$. After scanning all the paths, the FF table is shown in Figure 4. The FF of e_3 is e_8, since the frequency of $\{e_3, e_8\}$ is 2, the frequency of $\{e_3, e_4\}$ is 1. The same rule goes with e_7 and e_4. During compression, if the algorithm finds a following road segment of a certain road segment is an FF, the algorithm notes it down and keeps on looking if the following road segment of this FF is also an FF. Whenever there is no more concatenated FFs, the algorithm compresses the path formed by concatenated FFs and returns a tuple $< s_{FFE}, \delta >$. The special code s_{FFE} is for activating FFE, and δ indicates the number of road segments compressed. Then, the algorithm records the road segment following the last frequent follower on the compressed path, and the continue the above process. FFE runs in $O(|e|)$, same as SPE. It should be noticed that a path formed by frequent followers is different from a frequent path since each frequent follower is generated regardless of prefix.

4) Hotspot Frequent Compressive Path Enhancement: FCPE is a new algorithm proposed in this paper that leverages the idea of frequent pattern mining in compression scheme. It is known that pattern mining techniques [15] [16] can be utilized to reduce the redundant information on trajectory data. In terms of compression, pattern mining should not only focus on the frequencies of the sub paths, but also the lengths, since both frequency and length influence the performance of compression. Thus, we propose a function Φ for finding Frequent Compressive Path (FCP) suitable for compression. Both frequencies and lengths of the sub paths are taken into consideration. Moreover, if we store the FCPs

Algorithm 1 Hot Intersection Detection and FCP Mining

Require: Trajectory dataset $T(T_1, T_2, ..., T_n)$, Hotspot threshold b_{HOT}, Sub-path length bound b_{length}, FCP bound k.

Ensure: Hotspots v_{HOT} and a set of corresponding FCPs for every hotspot P_{final}.

```
v_S ← Get_Hot_Intersections(T, b_HOT)
for each v in v_S do
    Root(FP_tree) ← (v, 0)
    for each T_i in T do
        sub_path ← subTrajectory(T_i, v, b_length)
        Add(FP_tree, sub_path)
    for node N in FP_Tree do
        if NumberOfChildren(N) = 0 or NumberOfChildren (N) ≥ 2 then
            CandidateSet ← CandidateSet ⋃ (N, FCP_function(N))
    for 1 → k do
        Sort(CandidateSet)
        (N, FCP_value) ← RemoveFirst(S)
        P_FCP ← P_FCP ⋃ Path(v, N)
        for (N', FCP_function(N')) in CandidateSet do
            UpdatePrefix(N')
            RemoveRedundantValue(FCP_function(N'))
P_final ← P_final ⋃ (v, P_FCP);
```

into an universal table for compression, the long index of FCPs will hinder the compression performance. Due to the spatial locality of trajectories, FCPs can be stored relating to an intersection, FCPs starting at different intersections can share the same index with no conflict. FCPE utilizes a novel concept called hotspot. A hotspot is an intersection where the number of trajectories that pass through is larger than a threshold b_{HOT}. The frequent patterns starting at different hotspots will be stored separately, which significantly shortens the indexing overhead.

FCPE proceeds in two stages, FCP mining and FCP compression. FCP mining proceeds as follows: Firstly, the algorithm will scan through the sampled data to generate hotspots. Whenever a trajectory travels through or starts at an intersection, the frequency count of that intersection will increase by 1. Hotspots are intersections with frequency higher than a threshold b_{HOT}. Secondly, we vary a tree-based pattern mining method [17] to acquire the frequencies and lengths of all the sub paths starting at each hotspot. The sub paths rooting at the same hotspot will form a tree, where the frequencies and the lengths of all the sub paths and their shared prefixes can be mined. Though our FCP function can deal with various lengths and frequencies of sub paths, to bound the complexity of FPC mining, the lengths of sub paths are restricted by a bound b_{length}, which we set to 32. Finally, an FCP value is calculated for each mined sub-path by the FCP function Φ. The mined sub-paths are ranked by the FCP values. Top k FCP are selected and stored in an FCP table for each hotspot. We use pseudo code to provide the details of FCP mining, as shown in Algorithm 1.

The generation of FCP function is illustrated as follows: Consider a compression scheme $\mathbb{C}$ with average symbol size s_a for representing an individual road segment and the size of the symbol that activates frequent path compression s_{FCP}. Suppose a parameter k is set to generate top k paths. The lengths and the frequencies of paths are represented as x_n and c_n, respectively, where n is the total number of paths. We first present FCP function Φ by two parts, namely decompression time contribution Φ_t and compression ratio contribution Φ_s, as shown below:

$$\Phi(c_n, x_n) = W_s \Phi_s(c_n, x_n) + W_t \Phi_t(c_n, x_n), \quad (1)$$

where W_s is the weight of the compression ratio contribution and W_t is the weight of the decompression time contribution. In this paper, we focus on the compression ratio contribution of FCPs, thus W_s is set to 1 and W_t is set to 0.

First, we discuss decompression time contribution Φ_t. Frequent path mining based compression methods always include indexing different selected paths, where we can efficiently recover a whole path at once when such index exists. This means the more road segments the paths have, the more road segments there are to skip the individual searching, reading and writing than usual. Thus, we can calculate the total cost saved by storing a frequent path as:

$$\Phi_t(c_n, x_n) = c_n(x_n - 1)\eta, \quad (2)$$

where η is the actual cost of dealing with an individual road segment.

After discussing Φ_t, we move on to compression ratio contribution Φ_s. Recall the compression scheme $\mathbb{C}$, where the average size of symbols representing individual road segments is s_a and the size of the symbol that activates FCP is s_{FCP}. We can find that the longer the compressed path is, the more space is saved by representing more road segments using one FCP index. We can develop the compression ratio contribution of compressing one path as follows:

$$\Phi_s(c_n, x_n) = c_n(s_a x_n - s_{fcp} - \gamma(k)), \quad (3)$$

where $\gamma(k)$ is the size of the symbol for indexing top k FCPs. In our method, we use a binary fixed-length code to index FCPs, so $\gamma(k) = log_2 k$. The physical meaning of Φ_s is the storage space saved by compressing an FCP using the compressed represenation of FCP with frequency c_n.

The FCP fuction Φ is proposed by us. Next, we discuss the determination of k. Since we use a fixed-size index for indexing top k FCPs, a larger k may increase $\gamma(k)$ for every FCP in one table. Thus, the determination of k is a considerable problem. For scenarios that introduce different sizes of indexes for FCP, the influence of different k can be

well defined by Φ. Consider a fixed-size index for FCP, $\gamma(k)$ is decided by different k. Consider a fixed-size index as a sequence of symbols, $\gamma(k)$ will be influenced when we need one more symbol to represent all FCPs as k increases by 1. First, we want to know the size change ΔS of previous compressed FCPs if k is increased by 1, which can be expressed by the equation below:

$$\Delta S = \begin{cases} \sum_{n=1}^{k} \Phi_s(c_n, x_n) - \Phi_s^{'}(c_n, x_n), & \gamma(k+1) = \gamma(k) + 1, \\ 0, & \gamma(k+1) = \gamma(k), \end{cases}$$

where $\Phi_s^{'}$ is the new compression ratio conrtibution when $k = k + 1$. By solving the equation above, we can get:

$$\Delta S = \begin{cases} \sum_{n=1}^{k} c_n, & \gamma(k+1) = \gamma(k) + 1, \\ 0, & \gamma(k+1) = \gamma(k). \end{cases}$$

Thus, the condition of increasing k by 1 is:

$$\begin{cases} \sum_{n=k+1}^{k^*} \Phi_s(c_n, x_n) > \sum_{n=1}^{k} c_n, & \gamma(k+1) = \gamma(k) + 1, \\ none, & \gamma(k+1) = \gamma(k). \end{cases}$$

Since we use binary fixed-length representations for indexing k FCPs, we assume k varies among powers of two. The condition now changes to:

$$\begin{cases} \sum_{n=k+1}^{k^*} \Phi_s(c_n, x_n) > \sum_{n=1}^{k} c_n, & log_2(k+1) = log_2(k) + 1, \\ none, & log_2(k+1) = log_2(k). \end{cases}$$

The FCP compression stage is similar to that of SPE and FFE. When a hotspot is encountered during the compression of CE, the algorithm will keep tracking the following sub path from the hotspot to see if it matches any stored FCP. If the sub path matches more than one FCP with different length, the algorithm will choose the longer FCP to compress the sub path. A tuple $< s_{FCP}, \delta >$ will replace the original data. s_{FCP} is the special code for activating FCPE, while the δ is the index number of the stored FCP. The complexity of FCP compression is $O(|e|)$, where $|e|$ is the size of input.

5) Enhancements Coordination and Decompression: After introducing our hybrid map-matched path compression method, we discuss the coordination of ECE. As we have mentioned previously, each algorithm has its own way of functioning. In the real process, a logic called Session-based Enhancement Selector is introduced to ensure the coordination of these methods.

During the compression of ECE, CE will start compressing road segments, while SPE and FFE both denote the first road segment as datum point. The two enhancements will greedily find the longest consecutive straight path or frequent follower path. When one of them encounters a halt, i.e. no more road segments could be added to the straight path or frequent follower path beginning with the datum point, it will check the situation of the other enhancement. If the other one still continues, we will halt this enhancement and wait for the other one to halt. After both of the enhancements halts, ECE will select the one that compresses more edges. We keep this logic so that two enhancements start and end together to form a session. FCPE however, is different since a hotspot may occur in the middle of a session. However, this situation could be transfered to the situation of previous two enhancements by ending the previous session and starting a new session when a hotspot is spotted.

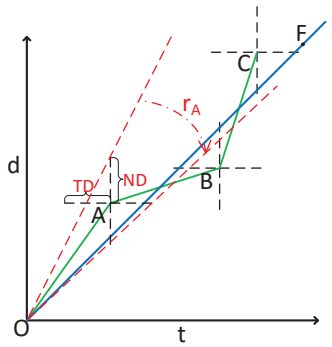

Figure 5: An example of FTS

The decompression of ECE is from the starting road segment to the end. We start deriving the road segments from the first road segment. Since their clockwise codes are embedded into the digital map, the decompresser can easily find related e_n upon receiving clockwise codes. When the special codes for SPE, FFE or FCPE are detected, we are able to derive e_n in a reverse way of encoding of the corresponding enhancements. The decompression yields a complexity of $O(|e|)$.

C. Temporal Compression

In this section, we introduce the temporal compression algorithm, namely Fitting-based Time Simplification(FTS). FTS is an improved algorithm based on BTC [10], which treats the (t, d) tuples as points to form a curve and adopt a line simplification algorithm to compress it. We adopt the error bounds presented in BTS, namely Network Synchronized Time Distance (NSTD) and Time Synchronized Network Distance (TSND) [10]. NSTD limits the error range of t, and TSND limits the error range of d. In this paper, we use TD and ND for short. We realize the improvement by introducing an approach similar to linear fitting. Different from previous line simplification algorithms that prune the points within error bound, our algorithm tries to fit an optimal line to represent the original curve. As shown in Figure 5, with the same error bound TD and ND, pruning method is not able to compress the curve $\{O, A, B, C\}$ due to the placements of the points, while fitting method can compress the curve $\{O, A, B, C\}$ to $\{O, F\}$

The compression procedure of FTS is as follows. We treat the first point of the curve as anchor p_{anchor}. We define an angular range $r_{overall}$ initialized as $(0, \frac{\pi}{2})$. Each next point p_n brings an angular range r_n calculated by the error bounds, as shown by A and r_A in Figure 5. The compressed line l_c must fall into the angular range in order to compress the corresponding points. Over a certain set of points, a compressed line l_c that can replace the points exists if the

angular ranges of the points have an overlap. Thus, each time, we update the overall angular range by intersecting the angular range r_n of the current point with the former angular range $r_{overall}$. When a point p_x with an r_x not intersecting $r_{overall}$ is detected, we return the compressed line by storing the anchor p_{anchor} and the slope s_{anchor} calculated by the final $r_{overall}$. Then, the algorithm will anchor at the point before p'. $r_{overall}$ is then initialized as $(0, \frac{\pi}{2})$. The process starts again. The concatenated l_c represented by p_{anchor} and s_{anchor} can produce an intersection i_n, which is not necessarily a point in the original data. We use i_n in the compressed curve. However, by calculating i_n, a part of the compressed point may fall out of error bounded range due to the partial change of l_c. Consider two concatenated l_c and $l_c{}'$, we deal with the issue by comparing t of $i_n(t, d)$ with t' of $p(t', d')$, where p is the point estimated by l_c. For the case of i_n being at the left of the second anchor, if $t > t'$, then no influence is introduced by i_n; if $t < t'$ for some p, we will re-estimate the error of such points. This goes the reverse way when i_n is at the right of the second anchor. If the error exceeded TD or ND, instead of reserving i_n, we keep an extra point $i_n{}'$ together with the $p_{anchor}{}'$ of $l_c{}'$ to avoid mistakes. $i_n{}'$ is calculated using the point $p(t, d)$ before p'_{anchor}, adapting its t to l_c or d to l_c according to different s_{anchor} of l_c. The complexity of FTS is $O(|\mathfrak{t}|)$, where $|\mathfrak{t}|$ is the number of the input timestamps.

D. Query Capability

As mentioned, a typical feature of map-matched trajectory compression is the capability of supporting queries without full decompression, thus decreasing the query overhead. In this paper, we introduce three basic queries, which can be extended to more complicated queries, namely $whereat$, $whenat$ and $intersect$. We will briefly demonstrate how our methods deal with these queries, and study the performances in Section V. We introduce the three queries as follows:

1 **whereat**: Given a trajectory T, a timestamp t as input, the query returns a triple $< e_n, d, ND >$, where e_n is the resulting road segment, d is the distance deviation from the starting point of e_n indicating the specific position, ND is the estimation error bound of the result.

2 **whenat**: Given a trajectory T, a position (x, y) on the trajectory, the query returns a tuple $< t, TD >$, where t is the resulting time of the given position, and TD is the estimation error bound.

3 **intersect**: This is a boolean query that returns $True$ if a trajectroy T has intersected a convex polygon P during a given time period (t_1, t_2).

To speed up the query process, we store some auxiliary structures to support the queries. For each road segment in the FF table of FFE, we store the real distances, the destinations and the minimum bounding rectangles (MBR) of the frequent follower paths with various lengths, up to 32. For SPE, we store similar structures. Also, we store the real distances and MBRs of the FCPs into the FCP table, and distance and MBRs for CE encoded road segments.

We will first illustrate how our query executer deal with $whereat$ queries. Given a $whereat$ query and a corresponding trajectory T. CCF will locate the timestamp t using compressed temporal data by finding a closest pair of compressed timestamps $< \mathfrak{t}_\alpha, \mathfrak{t}_\beta >$ that covers it. Using (t_α, d_α) and (t_β, d_β), the estimated traveled distance d_e of t will be fast retrieved. By adding up accumulative distance d_a from the start of the compressed spatial trajectory, we can derive the corresponding e_n and deviation distance d when d_a reaches d_e. ND is simply the error bound ND we use in FTS.

$whenat$ queries are handled in a similar way. For a given position (x, y) and a compressed trajectory T', the query executer keeps adding up accumulative distance and checking if (x, y) is on any part of the trajectory. CCF is able to skip extracting compressed trajectory by comparing MBRs with (x, y). Only when (x, y) is within an MBR that we need to extract the corresponding MBR and find if (x, y) is on any road segment inside. The distance d_a is found when (x, y) is reached. Then, with d_a we get, the targeting time t can be easily derived by the compressed temporal data. TD is simply the TD in FTS.

To deal with $intersect$, we first derive the distance pair (d_1, d_2) using the given timestamp pair (t_1, t_2). Then, we are able to locate a compressed sub-trajectory T^* using the given distance pair (d_1, d_2). After this step, we are able to efficiently compare if any MBR in T^* intersects P. If so, we further extract the original sub-trajectory corresponding to the intersected MBR to see if the decompressed sub-trajectory intersects P. If so, $True$ is returned, otherwise $False$ is returned.

The auxiliary structures for query acceleration bring some extra storage overhead. Since the size of the auxiliary structures are bounded by the size of the road network, they will not scale with input. These structures will only provide extra storage of 173.7MB, 173.7MB, 106.4MB, and 4.53MB, maximally. Under the circumstance of big trajectory data compression, we consider the sizes of such auxiliary structures tolerable to be ignored.

V. Experiment Evaluation

We report the experimental performance of CCF in this section.

A. Experiment Setup

We conduct experiments over the real trajectory data produced by 33,137 taxis of Beijing. The road network of Beijing has 226,237 road segments and 166,304 intersections. The number of trajectories is 47486, the length of trajectories sums up to 100,663,296 road segments. The experiments are conducted using a desktop with Intel i7 CPU and 64GB memory.

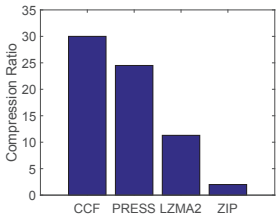

Figure 6: Compression Ratio Of Spatial Compression

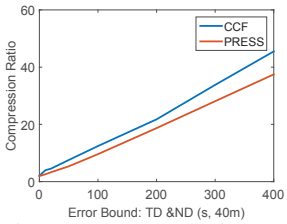

Figure 7: Compression Ratio Of Temporal Compression

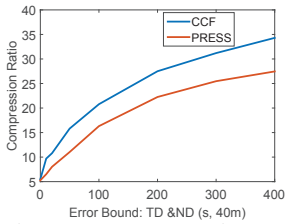

Figure 8: Compression Ratio Of Overall Compression

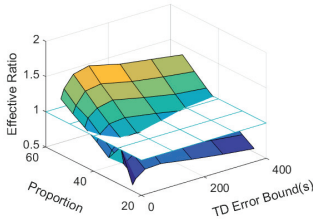

Figure 9: Effective Ratio of Different Proportions of Error Bounds

B. Metrics and Algorithms

We test CCF for its overall performance in terms of compression ratio, compression time, and query overhead. Compression ratio and compression time will be further studied in both spatial and temporal dimension. Since query performance involves both spatial and temporal compression algorithms, we will not study them separately.

Beside the algorithms in CCF, we also implemented the state-of-the-art compression algorithm, namely PRESS [10], for comparison. Since the spatial compression algorithm of CCF is lossless, we also test its performance against standard lossless text compression algorithms, such as LZMA2 and ZIP.

Before conducting the experiments, we use the map-matching algorithm in [18] to map the raw trajectories on to the road network. Thus, the input of our experiments is map-matched trajectory, which is represented by a map-matched path and a series of timestamps as defined in Section III. Due to the accuracy and sampling rate of positioning devices, the map-matched path can have some unconnected gaps inside. Current methods link these gaps with shortest paths for simplicity, which is not the real-life case, and it can boost the performance of shortest path related methods. CCF has a more stable performance since no shortest-path related method is included, while other methods may experience a drop in performance when a more reasonable route recovery algorithm is designed.

C. Compression Ratio

In this section, we evaluate the compression ratio of the proposed algorithms.

We first present the compression ratio of CCF's spatial compression algorithm, ECE. ECE is evaluated against the spatial compression algorithm of PRESS, namely HSC, and two other lossless text compression algorithms, LZMA2 and ZIP. The compression ratio of spatial compression is defined as $\frac{P_o}{P_c}$, where P_o is the size of original map-matched spatial paths, and P_c is the size of the compressed paths. The result is shown in Figure 6. ECE is the best among the four methods. ECE has the compression ratio of 30.2. PRESS achieves the second highest compression ratio of 24.5. Traditional text compression methods, namely LZMA2 and ZIP, achieve the compression ratio of 11.3 and 2, respectively.

Before reporting temporal and overall compression ratio, we conduct an experiment on the reasonable error-bound selection. To test our algorithms over various error bounds, we want TD and ND to have equivalent influence when compressing data. During the update of angular range $r_{overall}$ in FTS, a point p_n will produce four candidate boundaries of the angular range r_n, using TD and ND. Then two boundaries will be selected by r_n. We want to know the pairs of error bounds that make TD and ND range boundaries equally selected. Thus, we test various combinations of TD and ND and find the proportion of TD and ND that results in close times of taking effect during angular range update. Proportion is defined as $\frac{ND}{TD}$, and the axis TD is the value of TD. For example, if $TD = 10$ and $Proportion = 20$, it means $TD = 10$ and $ND = 200$. The effective ratio is presented as $\frac{c_n}{c_t}$, where c_n and c_t are the number of times that boundaries generated by ND and TD take effect, respectively. The experimental result is shown in Figure 9. The effective ratio is close to 1 when the proportion is around 40. This means both error bounds take effect evenly when ND is 40 times the value of TD. Thus, we will adopt this proportion for further experiments.

Now, we demonstrate the temporal and overall compression ratio of CCF. The compression ratio for temporal compression is defined as $\frac{\mathfrak{t}_o}{\mathfrak{t}_c}$, where $\mathfrak{t}_o$ is the size of original timestamps, and $\mathfrak{t}_c$ is the size of compressed timestamps. The compression ratio for overall compression is defined as $\frac{T_o}{T_c}$, where T_o is the size of the original map-matched trajectories, and T_c is the size of compressed trajectories. Since FTS is error bounded, we test the temporal and overall compression performance of CCF over various errror bounds. As shown in Figure 7 and Figure 8, CCF outperforms PRESS in various error bounds, both in temporal compression and overall compression. In terms of temporal compression, CCF outperforms PRESS steadily among different error bounds. In terms of overall compression, the compression ratios of CCF and PRESS are low when error bound is low. This is because the poor temporal compression ratio under low error bounds becomes the bottleneck of overall compression ratio. Overall, CCF outperforms PRESS in terms of compression ratio.

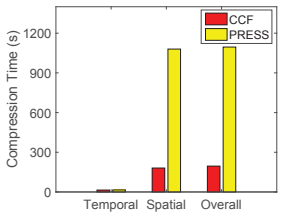

Figure 10: Compression Efficiency Comparison

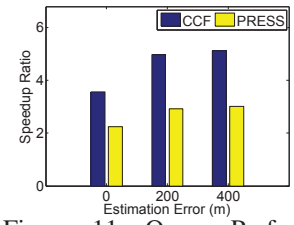

Figure 11: Query Performance of $whereat$

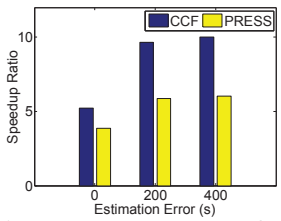

Figure 12: Query Performance of $whenat$

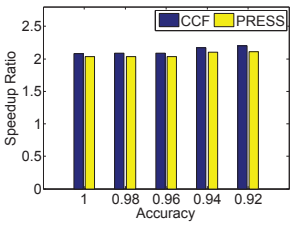

Figure 13: Query Performance of $intersect$

The compression of CCF requires some auxiliary structures, which are the extra part of CE preprocessed digital map, SPE table, FF table and FCP table. We assume all the auxiliary structures are built in advance and can be used for a long time, with the assumption that the road network and movement pattern of trajectories of a region won't have big changes rapidly. It should be noted that these structures are static and do not scale with the input size. Actually, the preprocessing of CE will only introduce an extra space of 3.9MB on the digital map maximally, while the SPE table, FF table and FCP table have the size of only 1.8MB, 1.8MB and 41MB, respectively. Thus, we consider such auxiliary structures are tolerable.

D. Compression Efficiency

In this section, we evaluate the compression efficiency of CCF. The compression efficiency of spatial compression, temporal compression and overall compression are studied. We evaluate the compression efficiency of CCF by the compression time of it. We compare the compression efficiency of CCF with PRESS.

The compression time of spatial compression, temporal compression and overall compression of both CCF and PRESS are shown in Figure 10. Since the compression time of FTS and PRESS have no significant change among different error bounds, we report a single value for temporal compression when the error bound is 200 and 8000 for TD and ND respectively. In terms of temporal compression efficiency, CCF and PRESS achieve a close performance, and the temporal compression time is significantly lower than spatial comperssion time. CCF shows better performance than PRESS in spatial compression efficiency, which takes only 17.67% the time of PRESS. In terms of overall compression efficiency, CCF is also better, which outperforms PRESS by 561%. The relatively lower performance of PRESS is related to the auxiliary structures it reserves for compression. The size of shortest path table scales fast with the complexity of road networks. In terms of the digital map of Beijing, the last-edge shortest path table can take up 110.6GB space. Such a big table will hinder the efficiency and bring extra disk access time. The road network of mega cities like Beijing also results in a more complicated Huffman tree. Overall, CCF outperforms PRESS in terms of compression efficiency.

E. Query Performance

In this section, we provide the experimental results regarding spatial temporal queries. An important feature of trajectory compression is the capability of conducting query on the compressed data without full decompression. As for CCF, it can retrieve query results faster than that on the uncompressed data with the help of query accelerating auxiliary structures. We generate random queries of three different types, namely $whereat$, $whenat$ and $intersect$ to test the query efficiency of CCF. We report the performance of $whereat$ and $whenat$ over different ND and TS, respectively. The performance of $intersect$ will be reported by with different accuracies brought by TD and ND. We use speedup ratio to denote the performance of CCF, which is represented as $\frac{t_o}{t_c}$. t_c is the time to process a query on compressed data, while t_o is the time to process a query on the original data. Still, we compare the performance of CCF against PRESS. The higher the speedup ratio, the faster the queries are. Due to the small size of CCF's query accelerating auxiliary structure, CCF fully supports in-memory querying and requires no disk access time for querying, while PRESS requires much I/O time due to the gigantic size of its query accelerating structure, which will dominate the query time since querying is a fast process. For direct comparison of querying capability, we exclude such time. The querying efficiency of PRESS will drop significantly when such time is considered. The results of $whereat$, $whenat$ and $intersect$ are shown in Figure 11 to Figure 13. We can see that CCF outperforms PRESS in terms of $whereat$ and $whenat$ query. In terms of $intersect$, the two approaches have a close performance. Such results prove the utility of CCF.

F. On-line Compression

We conduct an experiment simulating on-line trajectory compression. We maintain 1,000,000 trajectories with length of 50 road segments to update their location information in stream, with the time interval of 30s. Every 30s, each trajectory updates 5 new road segments and a new timestamp (In real-life case, the number of road segments may be less, here we use 5 to test our capability). In on-line case, CCF's three enhancements may not finish in one update, e.g. there are possible longer paths for FFE or FCPE. Thus, we maintain buffers for enhancements and update the result

Table II: On-line Compression Performance

Method	Spatial Compression Ratio	Latency
CCF	29	0.6364
PRESS	22(extended time)	N/A

of CE first. The verification of three enhancements can be finished during the rest of the time interval. When a coordinate session is complete and all enhancements finish computing, the result can be updated immediately in the next update. Even when not finished, the verified paths by enhancements can be immediately updated when the compressed data is occasionaly needed. We compare CCF with PRESS. Both CCF and PRESS are trained using a historical dataset with the size same as the size of the data to be compressed. Since temporal compression ratio is not influenced by on-line compression, we report the on-line spatial compression ratio and latency for CCF and PRESS, respectively. As shown in Table II, CCF uses only 0.6364 seconds to deal with each update, and the spatial compression ratio is 29. However, due to the fact that the gigantic shortest path table cannot be put into main memory at once, together with other reasons like the visiting time of the SP table and Huffman tree generated using a complex road network, PRESS is unable to deal with an update in time. We extend time until PRESS finish all the compression. The spatial compression ratio is 22. The results prove the on-line capability of CCF.

VI. Conclusion

In this paper, we propose CCF, a compression framework for trajectories under road network constraints. CCF employs novel algorithms to compress the spatial and temporal information of the trajectories. We test the performance of CCF using real dataset. CCF outperforms the state-of-the-art compression framework in terms of compression ratio, compression efficiency and query overhead. Moreover, CCF is able to compress on-line data streams. In the future, we plan to extend our framework to a wider range of queries, which makes the utility of the framework better.

VII. Acknowledgment

This research was supported in part by the University of Macau Grant SRG2015-00050-FST, NSFC grant 61300031, the National Key Basic Research and Development Program of China (973) Grant 2014CB340303.

References

[1] J. Owusu, F. Afukaar, and B. Prah, "Urban traffic speed management: The use of gps/gis," in *Conference proceeding, Shaping the Change XXIII FIG Congress Munich, Germany*, 2006.

[2] J. Grengs, X. Wang, and L. Kostyniuk, "Using gps data to understand driving behavior," *Journal of Urban Technology*, vol. 15, no. 2, pp. 33–53, 2008.

[3] C. Sung, D. Feldman, and D. Rus, "Trajectory clustering for motion prediction," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 1547–1552.

[4] D. Bernstein and A. Kornhauser, "An introduction to map matching for personal navigation assistants," 1998.

[5] G. Wang and R. Zimmermann, "Eddy: an error-bounded delay-bounded real-time map matching algorithm using hmm and online viterbi decoder," in *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2014, pp. 33–42.

[6] M. Veloso, S. Phithakkitnukoon, and C. Bento, "Sensing urban mobility with taxi flow," in *Proceedings of the 3rd ACM SIGSPATIAL International Workshop on Location-Based Social Networks*. ACM, 2011, pp. 41–44.

[7] P. M. Lerin, D. Yamamoto, and N. Takahashi, "Encoding network–constrained travel trajectories using routing algorithms," *International Journal of Knowledge and Web Intelligence*, vol. 4, no. 1, pp. 34–49, 2013.

[8] G. Kellaris, N. Pelekis, and Y. Theodoridis, "Map-matched trajectory compression," *Journal of Systems and Software*, vol. 86, no. 6, pp. 1566–1579, 2013.

[9] H. Cao and O. Wolfson, "Nonmaterialized motion information in transport networks," in *Database Theory-ICDT 2005*. Springer, 2005, pp. 173–188.

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

[11] S.-T. Wu and M. R. G. Marquez, "A non-self-intersection douglas-peucker algorithm," in *Computer Graphics and Image Processing, 2003. SIBGRAPI 2003. XVI Brazilian Symposium on*. IEEE, 2003, pp. 60–66.

[12] N. Meratnia and R. De By, "A new perspective on trajectory compression techniques," in *Proc. ISPRS Commission II and IV, WG II/5, II/6, IV/1 and IV/2 Joint Workshop Spatial, Temporal and Multi-Dimensional Data Modelling and Analysis*, 2003.

[13] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, and R. Jurdak, "Bounded quadrant system: Error-bounded trajectory compression on the go," in *Data Engineering (ICDE), 2015 IEEE 31st International Conference on*. IEEE, 2015, pp. 987–998.

[14] Y. Ji, H. Liu, X. Liu, Y. Ding, and W. Luo, "A comparison of road-network-constrained trajectory compression methods," in *IEEE International Conference on Parallel and Distributed Systems*. IEEE, 2016.

[15] W. Luo, H. Tan, L. Chen, and L. M. Ni, "Finding time period-based most frequent path in big trajectory data," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 713–724.

[16] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *ACM Sigmod Record*, vol. 29, no. 2. ACM, 2000, pp. 1–12.

[17] J. Han, J. Pei, Y. Yin, and R. Mao, "Mining frequent patterns without candidate generation: A frequent-pattern tree approach," *Data mining and knowledge discovery*, vol. 8, no. 1, pp. 53–87, 2004.

[18] P. Newson and J. Krumm, "Hidden markov map matching through noise and sparseness," in *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*. ACM, 2009, pp. 336–343.