2016 IEEE 22nd International Conference on Parallel and Distributed Systems

A Comparison of Road-Network-Constrained Trajectory Compression Methods

Yudian Ji†, Hao Liu†, Xiaoying Liu†, Ye Ding†, Wuman Luo‡

† *The Hong Kong University of Science and Technology*
‡ *University of Macau*
{*yjiab, hliuag, xyliu, valency*}*@cse.ust.hk,*{*wumanluo*}*@umac.mo*

***Abstract*—The popularity of location-acquisition devices has led to a rapid increase in the amount of trajectory data collected. The large volume of trajectory data causes the difficulties of storing and processing the data. Various trajectory compression methods are therefore proposed to deal with these problems. In this paper, we overview the existing road-network-constrained trajectory compression methods and propose a novel classification based on the features leveraged by them. We also propose new methods that fill in the research blanks indicated by the classification. We conduct a thorough comparison among the existing and new road-network-constrained trajectory compression methods. The performances of the methods are studied via various metrics on real-world dataset. We make new discoveries regarding the performances and the scalability of existing methods, and provide guidelines of road-network-constrained trajectory compression for various scenarios.**

***Keywords*-Trajectory compression, road network, compression algorithm, moving object database, spatio-temporal data**

I. Introduction

The popularity of location-acquisition devices has led to the emergence of large amounts of trajectory data, which are collected and valued by giant firms, governments and research institutes because it is of great importance in many application scenarios, such as urban sensing [1], path recommendation [2], behavior analysis [3], etc. However, the rapidly growing scale of trajectory data has caused the crises of storage and communication, which raises the demand for trajectory compression.

So far, many trajectory compression methods have been proposed. They can be grouped into two categories. Methods in the first category [4][5][6][7] focus on the compression of raw trajectory data. Typically, a raw trajectory is usually represented by a series of location points and the corresponding timestamps. Methods in the second category [8][9][10][11] focus on an increasingly important kind of trajectory data, namely trajectory data under road network constraints. A typical road-network-constrained trajectory records the movements of an object in a road network. It can therefore be aligned with the road segments and vertices of the road network for more precise and concise representation using map-matching methods [12][13], as shown in Figure 1. This new form of representation plays an important role

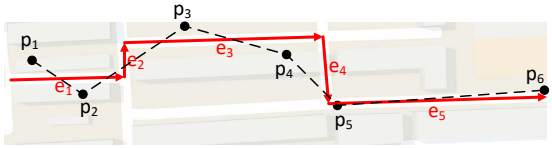


Figure 1: An Example of Map-matching

in trajectory-related applications. However, it introduces new challenges for trajectory data compression as well. In this paper, we focus on the comparison of the compression methods regarding road-network-constrained trajectories.

Various kinds of methods have been proposed to address the issue of road-network-constrained trajectory compression. MMTC [8] tries to replace some sub-paths of the map-matched trajectory with shortest paths with least information loss. Nonmaterial [9] aims at relating timestamps with road segments and using less timestamps to estimate the original ones within an error bound. Routing Algorithm [11] compresses trajectory data by removing the road segments that coincides some topological paths, such as shortest path and straight path. PRESS [10] adopts the shortest path compression same as that of Routing Algorithm, and develops an encoding algorithm to further compress the trajectory. It also modifies a line simplification algorithm for temporal compression.

Despite all the current work, there are still some problems to be solved. First, the experiments of existing work are not comprehensive enough. Not all methods have been tested against the same dataset under the same experimental setup. Moreover, they use either a trajectory dataset too small, or a road network too simple. The scalability of these methods are therefore unknown. Generally, a thorough comparison among the existing methods using a reasonably-sized dataset and a complex road network is needed. Second, so far, there is not a complete overview for road-network-constrained trajectory compression techniques, which brings difficulties in understanding the research topic completely. Existing methods leverage different features of road-network-constrained trajectory data. A complete overview gives good classification of the proposed work, and helps providing the future research direction. New methods could also be proposed to fill in the research blanks indicated by the classification. Thus, a comprehensive overview is required for road-network-constrained trajectory compression. Finally, there

1521-9097/16 $31.00 © 2016 IEEE
DOI 10.1109/ICPADS.2016.40

are opportunities for the existing work to borrow strong points from each other to solve their existing problems. For example, PRESS [10] proposes a novel encoding method as their final step of compression, which could be leveraged by other methods without encoding step to achieve better performance.

In this paper, we make a complete overview for road-network-constrained trajectory compression methods, and conduct a thorough experimental comparison among them. Specifically, we summarize the existing road-network-constrained trajectory compression methods by a novel classification. We also propose several new algorithms based on some novel observations from the classification. A thorough experimental comparison will be conducted among existing and new algorithms.

- We overview the existing road-network-constrained trajectory compression methods, and we propose a novel classification based on the features features leveraged by the compression methods. We further design several new spatial compression methods to fill the blanks of unexplored features in the classification, which are Most Frequent Follower Compression (MFFC), Most Frequent Path Compression (MFPC) and Frequent Path Compression (FPC).
- We conduct a thorough experimental comparison among existing and new road-network-constrained trajectory compression methods using a complex road network and real-world reasonably-sized trajectory dataset. We make new conclusions regarding the performances and scalability of existing methods.
- We come to other important conclusions as well. We give conclusions about the best combinations of methods under various performance metrics.

The rest of the paper is organized as follows. We make an overview that classifies the existing and new algorithms and introduce each of them in Section II. In Section III, we present a thorough experimental comparison among these algorithms. The conclusion is given in Section IV.

II. Algorithms

We introduce the existing and new algorithms in this section. We first present an overview of the methods, then the specific algorithms will be introduced in detail. Beside the existing methods, we also propose three new methods for spatial compression, namely MFFC, MFPC, and FPC.

A. Road-Network-Constrained Trajectory Representation

To better understand the road-network-constrained compression algorithms, we briefly describe the representation of road-network-constrained trajectories.

A road network G is a directed graph $G(V, E)$ that denotes the topology of real-world roads, where E is the set of road segments and V is the set of intersections. A path $P = \{e_1, e_2...e_n\}$ is a series of concatenated road segments.

When a trajectory T is under the constraints of a road network G, the location points $\{p_1, p_2...p_n\}$ can be aligned to the existing road segments $\{e_1, e_2...e_n\}$ in the road network G by map-matching algorithms [12][13][14]. T is then transfered to $T_{matched}$, which is the representation of road-network-constrained represented trajectory. $T_{matched}$ is represented by a path $P_{matched}$ indicating the spatial information, and a series of modified timestamps $\mathfrak{t} = (d, t)$ [10] indicating the temporal information. d is the traveled 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. The representation of timestamps can be different according to methods as long as it has no information loss. This difference will not affect the experimental comparison under the same setup.

B. Overview

As mentioned in Section II-A, road-network-constrained trajectory data contains two components, i.e. spatial information and temporal information. Obviously, a compression method can deal with spatial information, temporal information or both of them to achieve the aim of compression. Existing works [10][8][11] pay more attention on the compression of spatial information, since it is where the features of road network topology can be leveraged. More specifically, the spatial information is the new form brought by road networks and the major difference from other trajectories. The compression of temporal information is similar to time series compression and curve compression, where traditional text-based compression[15] and line simplification [4][5] methods can be easily applied. So far, only one dedicated new method [10] is proposed, which is based on an existing line simplification algorithm. We present overviews of both spatial and temporal compression methods, and give a classification of them.

Existing spatial compression methods can be classified from different aspects. In general, the methods can be classified into two categories based on the characteristics of road-network-constrained trajectory they leverage, namely topology-based and frequency-based. Topology-based methods compress trajectories purely based on the topology of road networks. In such case, no training or sensing on input data is needed. Frequency-based methods, on the other hand, capture the frequency information of historical data or input data for compression. A pre-training on the input or historical data is needed for such compression methods. Specifically, the methods can also be classified by the actual functioning logic of compression, which are simplification and encoding. Simplification methods remove some sub-paths from the original path, while encoding methods turn the path to bit strings. Simplification methods can be further classified into follower-based simplification and path-based simplification. Follower simplification methods focus on the relationship between one road segment and the next

Table I: Classification of Spatial Compression Methods

		Frequency-based	Topology-based
Path Simplification	Lossy	N/A	MMTC
	Lossless	*FPC, MFPC*	SPC
Follower Simplification	Lossless	*MFFC*	Follow-PA
Encoding	Lossless	FSTC	N/A

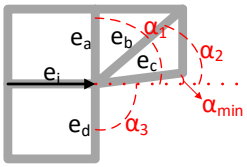
Table II: Classification of Temporal Compression Methods

General Methods	Lossy	Linear Interpolation Polygon Approximation
Dedicated New Methods	Lossy	BTC

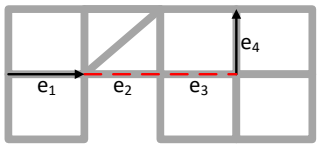
concatenated one. They simplify the original data by removing a series of concatenated followers with a certain local relationship between each two road segments. Path simplification methods aim at compressing a whole path that follows a certain rule (e.g. frequent path and shortest path). Such paths can be removed or replaced by shorter symbols from original data to achieve compression. Beside the above classifications, a method can also be classified by whether it is lossy or lossless, which denotes whether the decompression can fully recover the original data or not. It is assumed[10] that the alignment of map-matching methods are correct. Thus the lossy and lossless discussion does not consider the information change during map-matching.

We classify the existing methods and new methods we propose using the stated categories, as shown in Table I. Existing methods fall into different elements of the classification table. Looking at where the existing works lie in the classification, we can discover that some elements are unexplored. For example, there is no frequency-based path-based compression method yet. Thus, we propose new algorithms to fill some of the blanks of the classification, namely Most Frequent Follower Compression (MFFC), Most Frequent Path Compression (MFPC), and Frequent Path Compression (FPC). Unfortunately, we are not able to find any existing method or appropriate new method that falls into the element of 'frequency-based & lossy & path-simplification' or 'topology-based & encoding', which will be discussed as possible future works in Section IV.

In terms of temporal compression, there are three works who deal with temporal data. Nonmaterial [9] and MMTC [8] totally adopt general methods, such as linear interpolation and polygon approximation to either move the timestamps to intersections or use less timestamps to estimate multiple timestamps on the same road segment. These methods achieve limited compression ratio. PRESS [10] is the only existing work that proposes a dedicated temporal compression method for road-network-constrained trajectory compression, which is modified from a standard Before-Opening-Window [16] line simplification algorithm. So far, only lossy temporal compression methods are adopted. The classification of temporal compression algorithms is shown in Table II.



(a) Deviation Calculation



(b) Following Path Compression

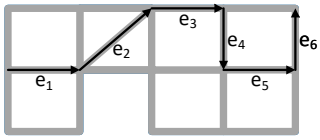
Figure 2: Compression Details of Follow-PA

C. Spatial Compression Algorithms

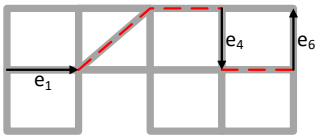
This section describes the spatial compression algorithms. We first give introduction to the existing algorithms, then we move on to the new algorithms proposed in this paper.

1) Follow-PA: Following Path Routing Algorithm [11] (Follow-PA) compresses spatial trajectories based on the assumption that the moving objects tend to follow a path with less deviation change when moving toward a destination. Intuitively, this algorithm removes the 'straight' paths in the original data. Consider a path P to be compressed, for each road segment e_i in P, the algorithm locates the set S_f of its possible following road segments $\{e_a, e_b, e_c, ...\}$ in the road network. Note that the next road segment e_{i+1} in P must be one of $\{e_a, e_b, e_c, ...\}$. The algorithm then calculates the angular deviation α_n from e_i to $\{e_a, e_b, e_c, ...\}$ respectively, as shown in Figure 2(a). If e_{i+1} has the least angular deviation α_{min} from e_i, e_{i+1} is denoted 'compressible'. Then the algorithm moves on to e_{i+1} and checks if e_{i+2} has the least deviation from e_{i+1}. The process repeats until the end of P. The road segments denoted 'compressible' are removed from P. An example is shown in Figure 2(b). Since e_2 is the following road segment of e_1 with least deviation, and same between e_3 and e_2, the path $P = \{e_1, e_2, e_3, e_5\}$ is compressed to $P' = \{e_1, e_4\}$. Follow-PA also considers the case of misalignment, which means even two road segments e_x and e_y do not share an endpoint, if e_y's starting point is within a small distance d from the ending point of e_x, e_y is still considered a possible following path of e_x. The decompression of Follow-PA is trivial, simply retrieving the series of following road segments with least deviation between unconnected road segments in the compressed path P'. Follow-PA yields the complexity of $O(|e|)$, where $|e|$ is the number of road segments in the input, since we encounter each road segment once.

2) SPC: Shortest Path Compression [11][10] (SPC) leverages the shortest path information of the road network to compress the spatial trajectory, under the assumption



(a) Original Path



(b) Compressed Path

Figure 3: Compression Details of Shortest Path Compression

that moving objects tend to follow shortest paths to reach their destinations. Generally, it removes a sub-path from the original path if it matches the shortest path with the same starting and ending points. It is assumed that the all-pair shortest path table is given. SPC anchors at the first road segment of an input path, and checks if the sub-path between the first and third road segment matches the corresponding shortest path. The examined sub-path is enlarged for one road segment for every positive feedback. SPC will remove the sub-path when a one-road-larger sub-path no longer matches the corresponding shortest path, and anchor at the road segment after the sub-path. The process repeats until the end of the input path. As shown in Figure 3, a path $P = \{e_1, e_2, e_3, e_4, e_5, e_6\}$ (Figure 3(a)) is compressed to $P' = \{e_1, e_4, e_6\}$ (Figure 3(b)), since $\{e_2, e_3\}$ matches the shortest path between e_1 and e_4, while $\{e_2, e_3, e_4\}$ no longer matches the shortest path between e_1 and e_5. The same goes between e_4 and e_6. The decompression of SPC is simply recovering shortest paths between unconnected road segments in the compressed path. SPC yields a complexity of $O(|e|)$, where $|e|$ is the size of input paths. (Shortest paths can also be computed on-demand, however, this brings the worst-case complexity of $O(|e||E| \log |V|)$, where $|V|$ and $|E|$ is the number of intersections and road segments in the road networks. This complexity is far worse than the shortest path version, especially in the complex road network of Beijing. Thus, we choose the shortest path version for comparison.)

3) MMTC: Map-matched Trajectory Compression [8] (MMTC) is a lossy compression algorithm that replaces certain sub-paths in the original path by point-to-point shortest paths. It assumes the all-pair shortest path table is given. MMTC uses an curve difference computation equation called MDL [8] to decide whether a sub-path should be replaced by the corresponding shortest path. The process of MMTC is as follows. The algorithm first anchors at the starting intersection of v_{1s} of e_1 of the original path $P = \{e_1, e_2, e_3 ... e_n\}$. For every next intersection, MMTC computes the MDL value according the sub-path formed by the two intersections and the corresponding point-to-point shortest path. The intersection v_{ie} with least MDL value is chosen, and the sub-path between v_{1s} and v_{ie} is then replaced by the corresponding shortest path. After that, MMTC anchors at v_{ie} and repeat the process. MMTC is a lossy compression algorithm with no decompression step. MMTC runs in $O(|e|^2 \log(|e|))$.

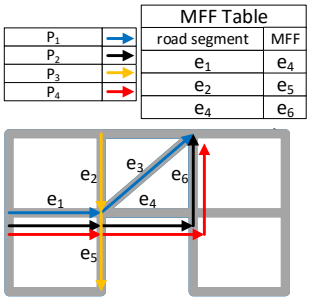


Figure 4: Most Frequent Follower Generation

4) FSTC: As an encoding algorithm, Frequent Sub-trajectory Compression [10] (FSTC) transfers original paths to Huffman codes. Generally, FSTC mines sub-paths from the training data within a length threshold θ together with their frequencies, and builds a Huffman tree using the sub-paths and their frequencies from the training data to replace the input data with Huffman codes. It scans through the training data and locates all sub-paths within the length of $\theta = 3$ [10]. Then, FSTC builds a prefix tree using the located sub-paths, where each node is one road segment, which can show up more than once in the tree. A path from the root to a node denotes a located sub-path. The weight regarding each node indicates the frequency of the sub-path denoted by the path from the root to this node. All undiscovered road segments in the road network are also included in the first level under the root in case some unseen sub-paths are to be compressed. With such prefix tree, a Huffman tree can be built by the nodes and their corresponding weights. The more frequent a sub-path is, the shorter Huffman code it receives. When compressing a new path, FSTC utilizes a logic similar to Aho-Corasick algorithm [17] to decompose the new path into Huffman coded sub-paths. The new trajectory is therefore encoded by Huffman codes. The decompression of FSTC is similar to Huffman decoding [15], i.e. simply recovering sub-paths by visiting the Huffman tree. The compression of FSTC runs in $O(|e|)$.

5) MFFC: Most Frequent Follower Compression (MFFC) is an algorithm proposed by us. It is frequency-based, follower-based and lossless. In real-world road networks, the traffic flow from one road segment to its possible following road segments can break down unevenly. One following road segment could be more frequently chosen than others by moving objects. From the above observation, MFFC compresses the path with the information of the most frequent follower. With a scan through the training data, we are able to discover the

following road segments for each road segment the data covers. Through a frequency statistics, a most frequent follower table is constructed to denote the most frequent following road segment for each road segment. We use an example to illustrate this. As shown in Figure 4, after scanning four paths $P_1 = \{e_1, e_3\}$, $P_2 = \{e_1, e_4, e_6\}$, $P_3 = \{e_2, e_5\}$, $P_4 = \{e_1, e_4, e_6\}$, a most frequent follower table is constructed. e_1's most frequent follower is e_4, since its follower e_3 has the frequency of 1, and e_4 has the frequency of 2. The same goes with e_2 and e_4. The compression stage of MFFC is similar to that of Follow-PA. During compression, MFFC starts from the first road segment of the input path P, and check if the second road segment is the most frequent follower of the first one. For every next road segment, MFFC checks if the most frequent follower is the next road segment in P. The process continues to the end of P. The road segments which are most frequent followers are then removed. The decompression of MFFC is to recover the removed most frequent followers by visiting the most frequent follower table. The complexity of MFFC is $O(|e|)$, where $|e|$ is the total size of input paths, since each road segment is visited once during compression and decompression.

6) FPC: Frequent Pattern Compression (FPC) is a new algorithm proposed in this paper. It is frequency-based, path-based and lossless. Intuitively, some paths in the road network are much more popular than other paths, such as the main streets of the city. These paths can be seen as frequent patterns. Similar to frequent pattern mining methods [18][19], we aim to mine the frequent patterns in the input data and compress them, as they are a typical kind of redundancy. FPC scans through the training data and locates all patterns with a fixed length l_{fp}. Since the length of the pattern is fixed, FPC simply has to locate a path with length l_{fp} starting from every road segment in the training data, except for some road segments at the tail of the paths, where the remaining path is not long enough. By counting the frequencies of the patterns, FPC creates a frequent path table that stores all fixed length patterns with frequency above the threshold s_{fp}. Each entry in the frequent pattern table stores an index and the corresponding frequent pattern. During compression, FPC starts looking for sub-paths that match the frequent patterns in the frequent pattern table from the first road segment of the input path. Whenever a matching sub-path is encountered, FPC replaces it with the corresponding index in the frequent pattern table. The decompression of FPC is also trivial, i.e. replacing the indexes by the corresponding frequent patterns. FPC runs in $O(|e|)$, where $|e|$ is the total size of input in edges.

7) MFPC: Most Frequent Path Compression (MFPC) is another algorithm proposed by us that leverages the idea of frequent pattern. Different from FPC, MFPC focus on the most frequent path between intersections in the road network. Similar to the logic of point-to-point shortest path, we want to compute point-to-point most frequent paths for each pair of intersections covered in the training data. However, to bound the complexity of the algorithm, we only compute point-to-point most frequent path with length no more than a threshold l_b. Similar to FPC and MFFC, MFPC scans through the training data to locate all sub-paths with length no more than l_b. Each sub-path has a starting intersection and an ending intersection. Different sub-paths can have the same starting and ending intersections. By statistics, a point-to-point most frequent path table is built to store the point-to-point most frequent patterns. Each entry stores a pair of intersections and the corresponding most frequent sub-path. The compression of MFPC proceeds as follows. MFPC anchors at the first road segment of the input path P and checks if the sub-path between the starting intersection of the anchored road segment e_a and the ending intersection of e_{a+l_b} matches the corresponding point-to-point most frequent path. If not, MFPC decrease the gap by checking e_a and e_{a+l_b-1}. We check points from far to near because unlike shortest path, most frequent path does not guarantee that if the prefix of a path P is the not most frequent path, P itself must not be the most frequent path. If a match occurs, MFPC removes the matched sub-path P_m from the input and continues to anchor at the next road segment not belonging to P_m. The process then continues the same way until the end of P. We reuse the example of SPC for description. As shown in Figure 3, suppose the MFP between e_1 and e_4 is $\{e_2, e_3\}$, and e_4 and e_6 e_5. A path $P = \{e_1, e_2, e_3, e_4, e_5, e_6\}$ (Figure 3(a)) is compressed to $P' = \{e_1, e_4, e_6\}$ (Figure 3(b)) by MFPC. The decompression is simply recovering MFPs by visiting the most frequent path table. The compression of MFPC runs in $O(l_b|e|)$, where $|e|$ is the size of input paths. In this paper, to balance compression ratio and efficiency, we set $l_b = 10$. Thus, the complexity turns to $O(|e|)$.

D. Temporal Compression Algorithms

Since only one dedicated algorithm for temporal compression is proposed, we give a brief description of it.

BTC: Since the temporal data of a map-matched trajectory is represented by (t, d) pairs, BTC [10] treats it as a function curve in $t - d$ space and modifies an error-bounded line simplification algorithm [16] and improve the complexity of it to linear time by introducing a concept called angular range. BTC presents both error bounds in dimension t and d, which are NSTD and TSND [10]. This is to bound the errors of two most common spatio-temporal queries, namely $Whereat$ and $Whenat$, respectively. Generally, BTC removes some points within the error bounds from the curve formed by (t, d) pairs, replacing them by straight lines formed by reserved points. The difference between the original curve l and the simplified curve l' is within the error bounds.

III. EXPERIMENTS

A. Experiment Setup

The device we use for experimental comparison is a One Amazon EC2 r3.xlarge instance server with Intel E5-2670 CPU and 32GB memory. Experiments are conducted using the real taxi trajectory data of Beijing. The road network of Beijing contains 226237 road segments and 166304 intersections. We use the map-matching algorithm proposed in [12] to generate map-matched trajectories. The map-matched dataset contains 24390 trajectories. Each trajectory has around 3000 road segments in average. The size of the map-matched trajectories is 1.085GB. All of the compared algorithms are implemented in Java.

B. Comparison Metrics

We test the algorithms using three metrics, which are compression ratio, compression efficiency and query overhead. Since there is only one dedicated temporal compression method [10], we mainly compare the spatial compression methods. The compression ratio is represented as $\frac{T}{T'}$, where T is the size of map-matched paths, and T' is the size of compressed paths. Compression efficiency is defined by compression time t_c and decompression time t_d. In terms of query overhead, we assemble BTC with the spatial compression methods and conduct query. We use three common spatio-temporal queries for comparison , where we randomly generate queries for each category and test the query overhead. The query overhead is represented by the efficiency ratio r_q, where the most efficient one has $r_q = 1$, and the others have $r_q \leq 1$ depending on how much they are slower than the best one. Some methods[10] may store query-related auxiliary structures to boost the query efficiency. For simplicity and fairness, we keep no such structure when conducting experiments, which is sufficient for comparison. We describe the three common spatio-temporal queries used in query overhead comparison as follows.

- **`GetTrajectory`**: Given a trajectory ID TID, this query returns the corresponding decompressed trajectory T.
- **`Whereat`**: Given a trajectory T, a timestamp t, the query returns $< e_n, \theta >$, where e_n is the resulting road segment, and the offset θ is the distance indicating the exact location of t on e_n from the starting point of e_n.
- **`Whenat`**: Given a trajectory T, a position (e_n, θ) as illustrated, the query returns the corresponding time t.

C. Preliminaries

1) Compared Algorithms: Among the introduced algorithms, a key observation is that FSTC [10] is an encoding algorithm, while the other algorithms are simplification algorithm, which means FSTC is orthogonal to other methods. Thus, beside the comparison of individual algorithms, we also combine FSTC with the other algorithms and conduct experimental comparison.

We propose experimental comparison among all described algorithms except MMTC. Firstly, MMTC is lossy and guarantees no error bound, while all the other algorithms are lossless. It is not appropriate to compare it with other algorithms under the same metrics. Secondly, the performance comparison of MMTC and the state-of-the-art methods is already done in [10], where MMTC shows limited performance and is greatly outperformed by PRESS. Since we focus on unsolved problems but not duplicating existing works, we exclude MMTC in comparison

2) Unconnected Gap Recovery: Due to the accuracy and sampling rate of the location acquisition devices, the map-matched path of a trajectory may sometimes contain several unconnected gaps within the path. Existing map-matching algorithms link these gaps with shortest paths for simplicity, which is what we do in this paper. However, the actual path between these gaps can be very different from the links. More importantly, such links will unfairly boost the compression ratio of shortest-path-related algorithms not only because they themselves are shortest paths, but also because they can link a long shortest path before their heads and after their tails. The compression performance of shortest-path-related algorithms could see a drop if the unconnected gaps are linked by frequency or probability based paths, which is closer to the real paths the moving objects travel.

3) Auxiliary Structures: The introduced algorithms maintain different auxiliary structures during compression, e.g. the frequent follower table of MFFC. Such structures are usually ignored since they do not scale with input size. According to our experiments, the size of the auxiliary structures of MFFC, FPC, FSTC and MFPC are reasonable, ranging from 1MB to 261MB. However, due to the complex road network of Beijing, the shortest path table of SPC has the size of 104GB, which is huge. This indicates that the shortest-path-related algorithms suffer a poor auxiliary structure storage scalability in terms of the complexity of road networks.

4) FPC and MFPC Configuration: Since FPC and MFPC have input parameters, namely the fixed length l_{fp} for FPC and the length bound l_b for MFPC. We conduct experiments using various l_{fp} and l_b to test the performance of FPC and MFPC, respectively. AS shown in Figure 9 and 10, to guarantee a good compression ratio as well as saving compression time, we choose l_{fp}=6 and l_b=10, respectively.

D. Compression Ratio Comparison

We first report the compression ratio of individual algorithms. As shown in Figure 5, SPC shows the best performance in terms of compression ratio, which is 8.62. MFPC and MFFC have close compression ratios to the best, which are 7.31 and 5.82, respectively. FSTC and Follow-PA

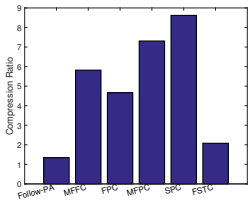


Figure 5: Compression Ratio of Individual Algorithms

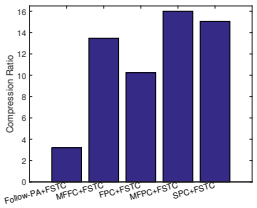


Figure 6: Compression Ratio of Combined Algorithms

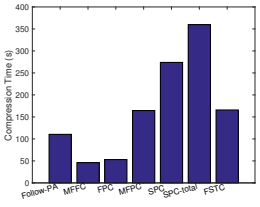


Figure 7: Compression Time of Individual Algorithms

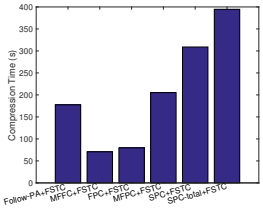


Figure 8: Compression Time of Combined Algorithms

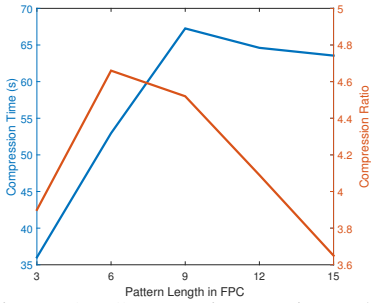


Figure 9: Compression Ratio and Time of FPC with Different l_{fp}

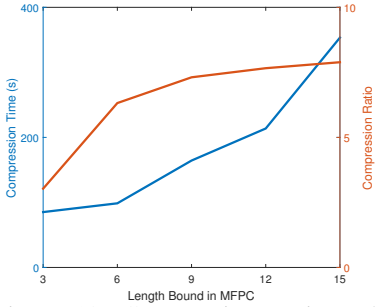


Figure 10: Compression Ratio and Time of MFFC with Different l_b

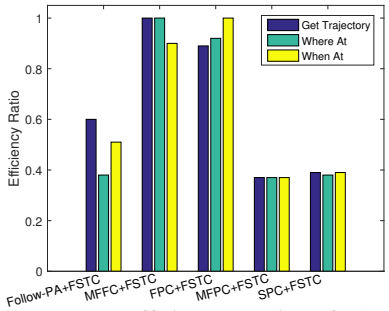


Figure 11: Efficiency Ratio of Spatio-temporal Queries (Higher is better)

show relatively poor compression ratios, while Follow-PA is the worst, which is 1.34.

Next, we report the compression ratio of combined algorithms. Each path or follower simplification algorithm is combined with FSTC. It is seen from Figure 6 that MFPC+FSTC is the best one, with the compression ratio of 16.00. MFFC+FSTC and SPC+FSTC show promising compression ratios, which are 13.47 and 15.05, respectively. Follow-PA presents limited compression ratio, which is 3.20. We can see that combined algorithms achieve better compression ratios than individual ones.

E. Compression Time Comparison

We compare the compression time of individual algorithms, and the results are shown in Figure 7. It can be seen that MFFC achieves the best compression efficiency, whose compression time is 45.94s. FPC is also efficient with the compression time of 52.96s. However, Follow-PA, MFPC, FSTC and SPC are performing disappointingly. Follow-PA, FSTC and MFPC have the compression time of 110.41s, 165.85s and 164.38s, respectively, which are far slower than MFFC and FPC. Most unfortunately, SPC has the longest compression time much worse that all other algorithms, which is 273.96s. Due to the size of shortest path table, we have to cut it into parts and load each part into main memory a time. Thus, taking the loading time (85.73s) into consideration, the compression time of SPC goes even worse, which is 359.69s.

Now we move on to the compression time of combined algorithms. As shown in Figure 8, the ranking is similar to individual algorithms. MFFC+FSTC has the best efficiency with the compression time of 70.68s. FPC+FSTC still achieves close efficiency to MFFC+FSTC, and the efficiency of Follow-PA+FSTC and MFPC+FSTC are still relatively bad. SPC+FSTC is the worst with the compression time of 308.97s, not to mention the 394.7s counting the loading table time. It is obvious that combined algorithms consume more time than individual ones during compression.

F. Query Overhead Comparison

Since the individual algorithms and combined algorithms have similar results in terms of efficiency ratio, due to the space limitation, we only present the results of combined algorithms. Note that the efficiency ratios of individual and combined algorithms are similar, but combined algorithms generally consume more time when conducting queries.

In terms of $GetTrajectory$, as shown in Figure 11, the two algorithms with good query efficiency are MFFC (Here we omit '+FSTC') and FPC, whose efficiency ratios are 1 and 0.89, among which MFFC is better, taking up the first position. The efficiency ratio of Follow-PA is in the median, which is 0.60. MFPC and SPC have close efficiency ratios, which are 0.37 and 0.39, respectively. These two algorithms are significantly slower in terms of query, where MFPC is the worst, if we do not consider the loading table time. In terms of $Whereat$, The best performing algorithm is MFFC, while the worst performing algorithm is MFPC. Follow-PA also sees a significant drop in performance in terms of $Whereat$ compared with $GetTrajectory$. The situation of $Whenat$ is similar to $Whereat$ but a bit different. FPC becomes the best performing algorithm instead of MFFC.

IV. Conclusion

In this paper, we summarize existing road-network-constrained trajectory compression methods and give them a novel classification. We also propose three new compression algorithms, namely MFFC, FPC and MFPC to fill the unexplored categories according to the classification. We conduct a thorough experimental comparison of existing and new methods. From to the experiments, we discover the performances of individual and combined algorithms in terms of different performance metrics. In terms of individual algorithms, the one with best compression ratio is SPC, and the one with best compression efficiency is MFFC. In terms of combined algorithms, the one with best compression ratio is MFPC+FSTC, the one with best compression efficiency is MFFC+FSTC, and the ones with lowest query overhead are MFFC+FSTC, MFFC+FSTC and FPC+FSTC for $GetTrajectory$, $Whereat$ and $Whenat$, respectively. Generally, combined algorithms achieve higher compression ratios, at the cost of losing some compression and query efficiency. SPC and MFPC have excellent compression ratios, but suffers low compression and query efficiency, especially for SPC. MFPC+FSTC may be a proper choice when storage problem is addressed. MFFC has good overall performance both in compression ratio, compression efficiency and query overhead. Thus, MFFC and MFFC+FSTC are good choices when compression ratio, compression efficiency and querying utility are all valued. We also discover that the existing algorithm SPC suffers poor auxiliary structure storage scalability in terms of the complexity of the corresponding road network. The compression and query efficiency of SPC are also unsatisfactory in the case of complex road network.

The possible future works are 1)extending the comparsion to more situations, such as online compression, 2)proposing new algorithms that fall in the unexplored categories according to our classification, which are 'frequency-based & lossy & path-simplification' and 'topology-based & encoding', and 3)proposing novel dedicated temporal compression algorithms.

V. Acknowledgment

This research is supported in part by Hong Kong RGC Grant HKUST16207714, the University of Macau Grant SRG2015-00050-FST, the NSFC Grant No.61300031, and the National Key Basic Research and Development Program of China (973) Grant 2014CB340303.

References

[1] 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.

[2] 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 international conference on Management of data*. ACM, 2013, pp. 713–724.

[3] 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.

[4] 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: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

[5] N. Meratnia and A. Rolf, "Spatiotemporal compression techniques for moving point objects," in *Advances in Database Technology-EDBT 2004*. Springer, 2004, pp. 765–782.

[6] C. Long, R. C.-W. Wong, and H. Jagadish, "Direction-preserving trajectory simplification," *Proceedings of the VLDB Endowment*, vol. 6, no. 10, pp. 949–960, 2013.

[7] J. E. Hershberger and J. Snoeyink, *Speeding up the Douglas-Peucker line-simplification algorithm*. University of British Columbia, Department of Computer Science, 1992.

[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] P. M. Lerin, D. Yamamoto, and N. Takahashi, "Encoding travel traces by using road networks and routing algorithms," in *Intelligent Interactive Multimedia: Systems and Services*. Springer, 2012, pp. 233–243.

[12] 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.

[13] H. Yin and O. Wolfson, "A weight-based map matching method in moving objects databases," in *Scientific and Statistical Database Management, 2004. Proceedings. 16th International Conference on*. IEEE, 2004, pp. 437–438.

[14] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang, "Map-matching for low-sampling-rate gps trajectories," in *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*. ACM, 2009, pp. 352–361.

[15] D. A. Huffman *et al.*, "A method for the construction of minimum redundancy codes," *Proceedings of the IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.

[16] N. Meratnia and A. Rolf, "Spatiotemporal compression techniques for moving point objects," in *International Conference on Extending Database Technology*. Springer, 2004, pp. 765–782.

[17] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Communications of the ACM*, vol. 18, no. 6, pp. 333–340, 1975.

[18] R. C.-W. Wong and A. W.-C. Fu, "Mining top-k frequent itemsets from data streams," *Data Mining and Knowledge Discovery*, vol. 13, no. 2, pp. 193–217, 2006.

[19] J. Han, M. Kamber, and J. Pei, *Data mining: concepts and techniques*. Morgan kaufmann, 2006.