Compressing Large Scale Urban Trajectory Data

Kuien Liu[1], Yaguang Li[1], Jian Dai[1], Shuo Shang[2] and Kai Zheng[3]

[1] Institute of Software, The Chinese Academy of Sciences,Beijing 100190, China
[2] China University of Petroleum, Beijing 102249, China
[3] The University of Queensland, Brisbane QLD 4072, Australia
{kuien,yaguang,daijian}@nfs.iscas.ac.cn,jedi.shang@gmail.com, kevinz@itee.uq.edu.au

Abstract

With the increasing size of trajectory data generated by location-based services and applications which are built from inexpensive GPS-enabled devices in urban environments, the need for compressing large scale trajectories becomes obvious. This paper proposes a scalable urban trajectory compression scheme (SUTC) that can compress a set of trajectories collectively by exploiting common movement behaviors among the urban moving objects such as vehicles and smartphone users. SUTC exploits that urban objects moving in similar behaviors naturally, especially large-scale of human and vehicle which are moving constrained by some geographic context (e.g., road networks or routes). To exploit redundancy across a large set of trajectories, SUTC first transforms trajectory sequences from Euclidean space to network-constrained space and represents each trajectory with a sequence of symbolic positions in textual domain. Then, SUTC performs compression by encoding the symbolic sequences with general-purpose compression methods. The key challenge in this process is how to transform the trajectory data from spatio-temporal domain to textual domain without introducing unbounded error. We develop two strategies (i.e., velocity-based symbolization, and beacon-based symbolization) to enrich the symbol sequences and achieves high compression ratios by sacrificing a little bit the decoding accuracy. Besides, we also optimize the organization of trajectory data in order to adapt it to practical compression algorithms, and increase the efficiency of compressing processes. Our experiments on real large-scale trajectory datasets demonstrate the superiority and feasibility of the our proposed algorithms.

Categories and Subject Descriptors H.2.8 [*Database Applications*]: Spatial databases and GIS

General Terms Algorithms, Design

Keywords Data Compression, Spatio-temporal Trajectory Data

1. Introduction

In recent years, the volume of spatio-temporal trajectory data generated from urban environments has drastically increased due to the prevalence of inexpensive GPS-enabled devices (cellphones or in-vehicle navigation terminals) and explosive growth of location-based services (LBS) and applications [23]. According to the latest statistics from GSA's reports [1], 775,000 Apple Apps and 700,000 Android Apps have been developed up to Oct. 2013, with an estimated 40% of them collecting user location information. As a result, more and more smartphone users are appreciating or relying on the capabilities of LBS in their daily lives.

The enormous volumes of trajectory data can easily overwhelm existing LBS and navigation applications, bring new challenges in storing, transmitting and processing this data, which urges the need for general-purpose, scalable data compression technique for trajectories.

Several trajectory compression methods have been proposed in recent years [7, 8]. Almost all of them can be classified into the category of lossy-compression [17], with the idea that replaces several consecutive trajectory segments by a time-parameterized motion function (e.g., a linear model $l(t) = l_0 + v_0 \times (t - t_0)$) while maintaining an acceptable degree of deviation caused by simplification. For example several work use the Douglas-Peucker [5] linear simplification technique while others take the parameter of time into account [16] or preserve speed and heading information [18].

Unfortunately, few of them are applicable to large scale trajectory compression. That is because these works focus on compressing each single trajectory independently, and they can be essentially treated as solutions to the min-ε problem [3]. The redundancy exists not only in a single trace itself but also across a trajectory collection.

In practice, urban moving objects share substantial common travel behavior since their movements are usually constrained by some geographic context (e.g., road networks or routes) [4, 11, 19]. For example, employees may commute between home and office driving the same routes hundreds of times, and trajectories collected form citywide gradually form a series of popular routes between uptown and downtown.

The phenomenon that redundancy exists among multiple trajectories and trajectory segments in urban environments makes it possible for us to reduce the size of trajectory data further. Two lines of research are closely related to the concept of redundancy among a set of trajectories, i.e., trajectory similarity search [13] and trajectory pattern mining [14]. However, they have different focus and the proposed algorithms are usually time consuming which makes them hard to be applied in trajectory compression directly.

We propose a scalable compression technique dealing with large scale trajectories in a collective manner. To guarantee its usability in practical applications, there are several challenges to be addressed: 1) the compression process should be fast, 2) the compression error should be adjustable, and 3) the compress algorithm should be adaptive to resource-limited settings. We develop a method (SUTC) that addresses these challenges by transform-

CloudDP'14, April 13, 2014, Amsterdam, Netherlands.

http://dx.doi.org/10.1145/2592784.2592787

ing spatio-temporal trajectories into text sequences using different symbolization strategies, and allow most applications to adjust the restoration error within tolerant range. We also improve the trajectory data organization for compression to adapt to computing environments.

In general, the major advantages of our solution are that we are committed to capture the similarity among multiple trajectories in a highly efficient way, i.e., time complexity is $O(n)$, and reduce the redundancy information using most common compression methods, which have usually been integrated into operating systems and databases as a fundamental component. Benefiting from these attributes, SUTC works well in practical environments.

The main contributions of this paper can be summarized as follows:

- Taking into consideration the redundancy commonly existing among large-scale urban trajectories, we propose a working solution called scalable urban trajectory compression (SUTC).
- Proposing two symbolization strategies with help of underlaying road networks, we first transform spatio-temporal trajectories into textual sequences and then compress large sets of trajectories using general-purpose compression techniques.
- Reorganizing the trajectory data according to several practical factors other than maximum compression error, we can perform SUTC algorithms in various computing environments, e.g., with limited memory capacity.
- Performing an experimental study conducted over real trajectory datasets, we demonstrate the effectiveness and the efficiency of the proposed solution.

The remainder of this paper is organized as follows. In Section 2 we briefly introduce the related works. Section 3 presents the overview of our solution as well as two symbolization strategies, followed by improvement efforts toward practical environment setting in Section 3.4. Experimental results are discussed in Section 4. Finally, Section 5 concludes the paper.

2. Related Work

There is a vast body of literature on trajectory compression in recent years. Good surveys on trajectory compression are provided in [7, 8, 17]. Line simplification, the idea of which is borrowed from computer graphics and cartography, is well studied in area of trajectory compression. Most of the line simplification algorithms operate in the two dimensional Euclidean space. For example, in paper [16] proposes a compression technique that uses the Douglas-Peucker method (DP) [5] and takes the parameter of time into account. In particular, it replaces the Euclidean distance used in the DP method by a time-aware one, called Synchronous Euclidean Distance (SED). Besides, the STTrace algorithm [18] is designed to preserve heading and speed information together with spatial distance. This kind of techniques is easy to understand and implement and can be roughly considered as solutions to the min-ε problem [3], i.e., for a given number of vertices, find the simplified polygon chain with the lowest distance to the original polygonal.

Single trajectory compression can also be interpreted as finding out a motion function to approximate the original trajectory with acceptable deviation. The motion function can be described with linear or non-linear models. Line simplification algorithms [16, 18] is a linear model, which assumes an object moving with constant speed till the offset between actual positions and estimated position exceeds a deviation threshold. Besides, non-linear methods such as Chebychev polynomials [2], cubic spline [10] and acceleration profile are imported to approximate a trajectory. Inevitably, they always suffer from higher computational complexity than that of linear ones.

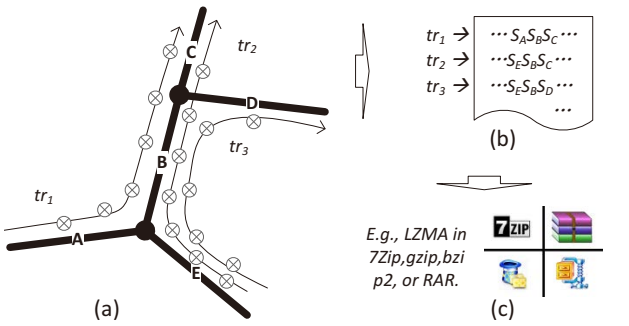

Figure 1. Processes of SUTC: (a) matching a set of trajectories onto the road network, (b) representing them using symbolization strategies, and (c) compressing symbolized trajectories using general-purpose methods.

An off-line compression method related to our work is the one by Cao and Wolfson [21]. However, our goal is to reduce the storage size of large-scale trajectory data without altering its infrastructure, but only by preprocessing and invoking common compression methods. Also related to ours are the work based on the priori knowledge of the road network, e.g., MMTC [9] explore the combination of the map-matching with the storage-space problem, STC [20] achieves its compression rate by replacing raw position information with a semantic representation of the trajectory consisting of a sequence of events, and work [6] proposes a solution that dilutes (or simplifies) the trajectories to routes by Map-matching and DP [5] and then codes routes over a vectorial road network. But all these works focus on encoding a single trajectory rather than all at once.

There are two kinds of research close to the concept of redundancy among a set of trajectories, i.e., trajectory similarity search [13] and trajectory pattern mining [14]. These works have certain value of reference to redundancy detection. However, their objective is to analyze the similar behavior of moving objects and always cause huge computational cost, whereas this paper is to extract a higher level of redundancy among large-scale trajectories.

Another related research topic is data compression in the database (e.g., light-weight database compression [22]), but it is designed to compress data of generic types using lossless scheme and, consequently, are not directly applicable here. Our approach is a lossy compression scheme which aims to utilize the reduced/compressed trajectories to get both higher storage capacity and faster response.

In general, the key difference between most of existing works and our work is that we focus on how to compress a large set of urban trajectories by general-purpose compression techniques.

3. Scalable Urban Trajectory Compression

3.1 Overview of SUTC Algorithm

Due to the constrained transportation infrastructures, such as underlying road network, urban trajectories often repeat themselves. Intuitively, although the number of urban trajectories can be huge, most of the movement routes on the road network in a city are relatively fixed. Hence, we can use the map-matched road segments to symbolize the original trajectories, and then perform compression on resulted strings.

Our approach consists of three steps as shown in Figure 1. First, we use a highly scalable map matching algorithm to associate the original trajectories with the road network [12, 15]; then, several novel symbolization strategies are adopted to convert the matched trajectories to plain strings; finally, conventional compression algo-

rithms and tools (such as the Lempel-Ziv-Markov chain algorithm (LZMA) in 7zip) can be directly applied to these plain strings.

The reasons why the symbolization strategies play an important role in trajectories compression are as follows.

- The positioning information in the original trajectories is represented in floating-point numbers. Hence, the detection of spatio-temporal duplicates is not that straightforward, which will result in the poor performance of conventional compression algorithms.
- Compared with the Chebyshev polynomials fitting and the spline interpolation, the symbolization technique is more efficient and scalable to deal with large scale urban trajectories.
- The symbolization technique is robust and easy to be combined with other compression algorithms and tools. Therefore, even if, in some cases, the trajectories have not been compressed sufficiently in step two, further compression can be made in the final step.
- We can customize the symbolization process to generate short and unified symbols.

Since the major component of SUTC is to convert the raw trajectories to symbolized sequences, we will elaborate two trajectory symbolization strategies in section 3.2 and section 3.3, respectively.

3.2 Velocity based Trajectory Symbolization($SUTC_v$)

In urban environments, the traffic conditions on certain road segments are usually stable within a certain period. Accordingly, the travelling velocities on them are relatively fixed. Inspired by this observation, we propose a velocity based trajectory symbolization strategy (SUTC$_v$).

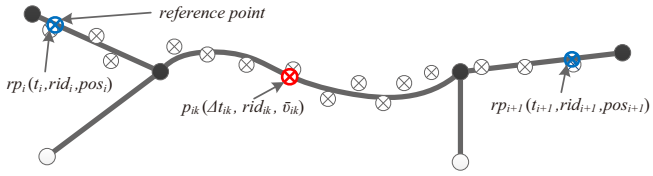

Figure 2. Velocity based Trajectory Symbolization (when the distance deviation is greater than the predefined δ, a new reference point will be generated).

Algorithm 1 shows the pseudo-code of this process. As a preliminary step, the algorithm performs the map-matching to convert the trajectory $((t_i, x_i, y_i)_{i=1}^n)$ to a map-matched sequence $(rp_i)_{i=1}^n = (t_i, rid_i, pos_i)$ (Line 2), where rid_i denotes the id of map matched road segment, pos_i denotes the distance between the point and the starting junction of the road segment. Figure 2 illustrates the map-matching process of the SUTC$_v$. As $p_1(t_1, x_1, y_1)$ is the first sampling point of the trajectory, it automatically becomes a reference point (Line 1).

Then the matched sequence is converted to $((rp_i, (sp_{i_j})_{j=1}^{n_i})_{i=1}^k)$ (Lines 4 - 20), where $rp_i = (t_i, rid_i, pos_i)$ denotes the i*th* reference point (matched onto the road network), and $sp_{i_j} = (\Delta t_{i_j}, rid_{i_j}, \overline{v}_{i_j})$ is the sliding point along the road network, in which Δt_{i_j} is the sampling interval between two consecutive samplings, rid_{i_j} is the matched road segment, and $\overline{v}_{i_j}$ is the average velocity between two consecutive samplings.

$$\widetilde{v} = \textbf{Round}\left(\frac{d(p_i, p_j)}{t \cdot \Delta v}\right) \cdot \Delta v \qquad t = \sum_{k=i+1}^{k \le j} \Delta t_k \qquad (1)$$

Algorithm 1 The Velocity based Trajectory Symbolization

Input:Road network G, trajectory $T = (t_i, lat_i, lng_i)_{i=1}^n$, distance deviation bound δ
Output:The compressed result: T_{str}

1: $T_{str} \leftarrow$ NULL, $i \leftarrow 2, d \leftarrow 0, \widetilde{d} \leftarrow 0, SP \leftarrow$ NULL
2: $RT \leftarrow$ **Mapmatching**(T) // $RT = (rp_i)_{i=1}^n = (t_i, rid_i, pos_i)_{i=1}^n$
3: $rp \leftarrow rp_1$
4: **while** $i \le n$ **do**
5: $\quad \Delta t \leftarrow p_i.t - p_{i-1}.t$
6: $\quad P \leftarrow$ **FindPath**(p_{i-1}, p_i)
7: $\quad \bar{v} \leftarrow$ **Length**$(P)/\Delta t$
8: $\quad \widetilde{\bar{v}} \leftarrow$ **Round**$(\bar{v}/\Delta v) \cdot \Delta v$
9: $\quad d \leftarrow d + \bar{v} \cdot \Delta t$
10: $\quad \widetilde{d} \leftarrow \widetilde{d} + \widetilde{\bar{v}} \cdot \Delta t$
11: $\quad$ **if** $|d - \widetilde{d}| < \delta$ **then**
12: $\quad\quad SP$.**Append**$((\Delta t, rid_i, \widetilde{\bar{v}}))$
13: $\quad$ **else**
14: $\quad\quad T_{str}$.**Append**$((rp, SP))$
15: $\quad\quad SP \leftarrow$ NULL, $rp \leftarrow rp_i$
16: $\quad\quad d \leftarrow 0, \widetilde{d} \leftarrow 0$
17: $\quad$ **end if**
18: $\quad i \leftarrow i + 1$
19: **end while**
20: T_{str}.**Append**$((rp, SP))$
21: **return** T_{str}

where $d(p_i, p_j)$ is the shortest distance between p_i and p_j, Δt_i is the time interval between t_i and its previous point.

In Line 6, function **FindPath** is used to find the shortest path from p_{i-1} to p_i. In Lines 7 - 8, the algorithm calculates the average speed and the rounded average speed between the two adjacent points. As shown in Equation (1), the $\bar{v}$ is rounded with the velocity interval Δv in order to improve the compression effectiveness.

In Lines 9 - 17, the algorithm calculates the distance deviation to determine whether a reference point should be created. If the deviation is smaller than δ (Line 11), the calculated sliding point $(\Delta t, rid_i, \widetilde{\bar{v}})$ will be appended to the sliding point sequence SP. Otherwise, the whole sliding point sequence together with the reference point will be appended to the result (Line 14), and then a new reference point will be created (Line 15).

3.3 Beacon based Trajectory Symbolization(SUTC$_b$)

In SUTC$_v$, the first point that results in intolerable deviation, i.e., deviation that is greater than δ, is treated as a new reference point. However, the efficiency of this algorithm tends to be less satisfactory when the number of trajectories grows larger due to the time-consuming shortest path finding algorithm.

Since the road networks are fixed, we can establish our reference points off-line and reuse them as beacons rather than computing from the scratch. Motivated by this idea, the beacon based trajectory symbolization algorithm (SUTC$_b$) is proposed, which is illustrated in figure 3.

In SUTC$_b$, we define the beacon distance d to split the road segment into fragments rfs. These fragments are labeled by adding a suffix to the original road segment id, i.e., rf_{idx}. For example, if we set $d = 100m$, and the length of the road segment with id 4562 is $281m$, then we can split it into three road fragments rf_{4562_1}, rf_{4562_2} and rf_{4562_3}, where the lengths of first two fragments are $100m$ and the length of the last one is $81m$. In general, Equation (2) is used to get the fragment indices.

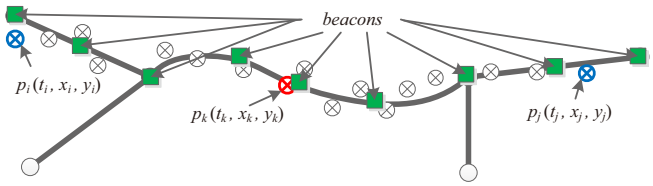

Figure 3. Beacon based Trajectory Symbolization (each junction automatically becomes a beacon, then the other beacons are computed in accordance with the Equation (2)).

$$[htbp]idx = \left\lfloor \frac{rs}{d} \right\rfloor \quad (2)$$

where rs denotes the distance between the starting junction of the road segment and the starting point of the fragment, while d is the beacon distance.

As the map matched sampling point is aligned to its nearest beacon, the maximum distance between the beacon and the sampling point is limited to half of the beacon distance.

Algorithm 2 The Beacon based Trajectory Symbolization

Input:Road network G, trajectory $T = (p_i)_{i=1}^{n} = (t_i, lat_i, lng_i)_{i=1}^{n}$, beacon distance d
Output:The compressed result: T_{str}

```
T_str ← NULL, i ← 1, t ← 0
while i ≤ n do
    rfid ← GetNearestBeacon(G, p_i)
    T_str.Append((p_i.t − t, rfid))
    t ← p_i.t, i ← i + 1
end while
return T_str
```

Another merit of SUTC$_b$ is that it can convert the original trajectories to strings without computing the shortest distance between two sampling points, which greatly reduces the compression time.

Algorithm 2 gives a brief description of SUTC$_b$. Function **GetNearestBeacon** is used to get the nearest beacon to the trajectory point. As beacons are indexed using Grid or R-Tree, this can be done in $O(log(N))$ time complexity, where N is the number of beacons in the road network.

3.4 Optimization based on data reorganization($SUTC^{+}_{[v|b]}$)

To provide symbolic sequences (or strings) which are more suitable to be compressed by conventional algorithms, we need to put similar trajectories together. Because for both SUTC$_b$ and SUTC$_v$, our approaches tend to symbolize the similar trajectories to the similar or identical strings. And the lossless data compression algorithms (such as Lempel-Ziv-Markov chain algorithm, LZMA) use dictionary scheme, which adopts a dynamic programming framework to find a near optimal compression solution through a *sliding window* that consists of two parts: *search buffer* and *look-ahead buffer*. The encoder in these algorithms tries to find the longest pattern in the *search buffer* that matches the pattern in the dictionary (or the *look-ahead buffer*). If a matched pattern is found, it is then encoded as the combination of offset, length and the next symbol. Therefore, when similar strings are organized adjacent to each other and proper compression algorithm configuration is used, we can expect to achieve the maximum compression ratio.

However, in practice, the size of the sliding window(the buffers) cannot be arbitrarily large due to the limited memory capacity of the encoder. Hence, rather than increase the memory size, we consider to reorganize the original data into a set of small partitions before conducting SUTC$_b$ and SUTC$_v$, where the size of each partition is similar to the size of sliding window. Three kinds of reorganization can be adopted: 1) temporal-based, which partitions trajectories according to specific time slots; 2) spatial-based, which partitions trajectories according to spatial space; and 3) hybrid, which considers both spatial and temporal domains. For example, one straightforward spatial-based strategy is uniform partition (i.e., grid). This paper will not list them with details due to limited space. In the experiment evaluation section, we will show such kind of partition can achieve near optimal result, especially when the size of trajectory data grows larger.

4. Experimental Study

In this section, we first give a brief description of the dataset and the experiment settings, then we evaluate the performance of proposed algorithms and compare it with a well-known algorithm, finally we discuss the effects of trajectory reorganization.

4.1 Dataset and Experimental Settings

The dataset used in the experiment was collected from 48,836 taxis in Beijing, China during a period of 7 days, and it contains about 500 million trajectory points. The corresponding road network contains 171,186 intersections and 452,474 roads.

In our experiments, preprocessing steps, such as map-matching and data-cleaning, have been done in advance. In addition, all the trajectories are matched to the road network using the map-matching algorithm in [12, 15]. We choose the Douglas-Peucker (DP) algorithm [16], which is the most representative work for trajectory compression, as the baseline algorithm. Without loss of generality, we also perform the well-known LZMA compression algorithm (implemented version in 7zip) on the symbolized trajectories. The parameters of LZMA are as follows, dictionary size: 64MB, word size: 32, solid block size: 2GB and thread number: 2.

All experiments are coded in C# and conducted on a computer that is equipped with 2.6GHz I5-3320M CPU and 8GB memory.

4.2 Effectiveness of SUTC Family

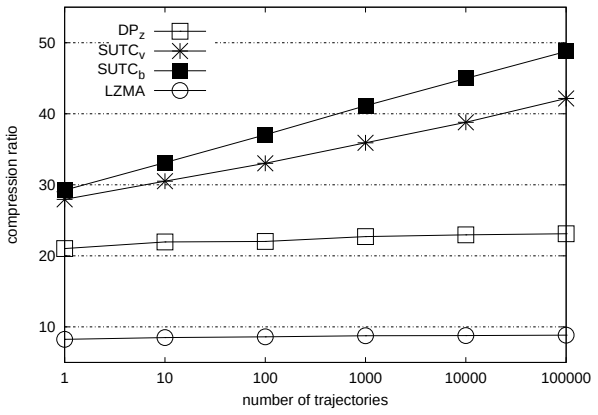

Figure 4. Compression ratio vs. Number of trajectories (maximum deviation $\delta = 30m$, sampling interval $\Delta t = 15s$).

To evaluate the effectiveness of proposed algorithms, we compare them with DP and LZMA algorithm. Figure 4 illustrates the compression ratios of SUTC$_v$ + LZMA (i.e., we first run the SUTC$_v$ symbolization algorithm and then perform LZMA on the output. We will refer to it as SUTC$_v$ for simplicity), SUTC$_b$ + LZMA (abbreviated SUTC$_b$ correspondingly), DP + LZMA (abbreviated DP$_z$), and LZMA with maximum deviation $\delta = 30m$,

sampling interval $\Delta t = 15s$ and vary the number of trajectories N from 1 to 100,000.

As the number of trajectories grows larger, the compression ratios of both SUTC$_v$ and SUTC$_b$ increase significantly, while the compression ratios of other algorithms (LZMA and DP$_z$) remain roughly the same. This is because that, with the number of the trajectories increases, similar patterns will appear more frequently. As these similar patterns are transformed to identical strings, the compression ratio will consequently increase. When the number of trajectories grows to 100,000, the compression ratio increases to 50:1 which is much larger than simple LZMA and DP$_z$. Therefore, SUTC algorithms are more suitable for compressing large scale trajectory data.

As indicated in Figure 4, the compression ratio of SUTC$_v$ is generally lower than SUTC$_b$. This is because in the urban area, taxis tend to have relatively low but frequently-changed speeds. Thus, the SUTC$_v$ algorithm are less likely to generate similar or identical records than the SUTC$_b$ algorithm. Moreover, SUTC$_v$ is supposed to have better performance when the moving objects are running on long roads with constant speeds, e.g., in the high way.

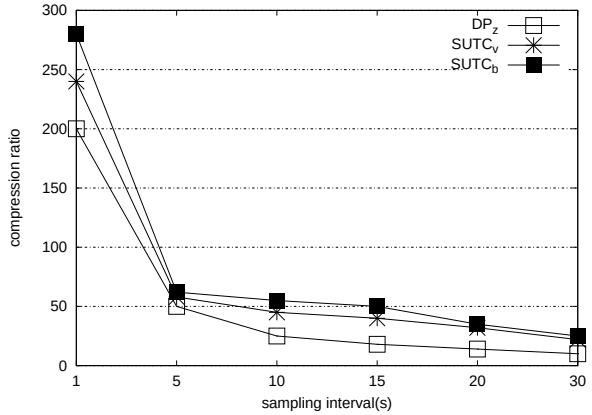

Figure 5. Compression ratio vs. Sampling interval ($N = 10,000, \delta = 30m$).

Figure 5 illustrates the relationship between the compression ratio and the sampling interval with $\delta = 30m$ and $N = 10,000$. Smaller sampling interval means more redundancy, i.e., there will be more sampling points in the same road with similar positions and velocities. The redundancy will finally be converted to similar or identical strings in SUTC$_v$ and SUTC$_b$. Consequently, the compression ratio will increase with the decrease of the sampling interval.

We also vary the value of the maximum deviation δ before/after symbolization from 5 meters to 200 meters to evaluate the performance of proposed algorithms. Figure 6 shows the effect of maximum allowed deviation: δ, with the number of trajectories $N = 10,000$. Larger error tolerance means fewer changes in the converted trajectory points, i.e., more identical string patterns and more redundancy. Although the size of the file generated by the SUTC symbolization algorithms will not decrease much, the LZMA algorithm will achieve a much better performance because of the redundancy. Hence, the compression ratio increases with the increase of δ. However, when δ continues to increase (exceeding 75m), which roughly equals to the half average length of the roads, the growth rate of the compression ratio becomes very low. This is because the symbolization strategies are relying on the road network, which can further optimized with some methods like [4]. To better understand this observation, we further study another attributes, average deviation.

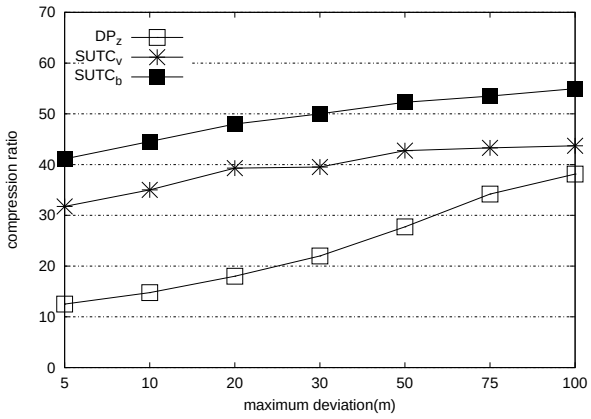

Figure 6. Compression vs. Maximum deviation ($N = 10,000, \Delta t = 15s$).

When δ becomes smaller, the compression ratios of all the algorithms decrease. Generally, the SUTC algorithms are less sensitive to δ which means these algorithms can achieve a good compression ratio while being highly accurate.

Figure 7 shows the average deviation of the compression algorithms. Although all the algorithms have the same maximum deviation, SUTC algorithms have much lower average deviations. This is because no matter how large the maximum deviation is, the SUTC algorithms will always keep the skeleton of the trajectory, i.e., the road sequence passed by the moving objects. Therefore, SUTC algorithm tend to be more accurate with identical maximum deviation.

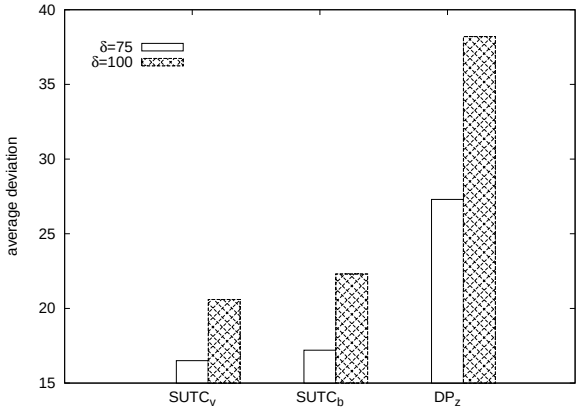

Figure 7. Average deviation ($N = 10,000, \Delta t = 15s$).

Figure 8 demonstrates the effects of trajectory reorganization with $N = 10,000, \delta = 30m$. In this experiment, we implement a straightforward spatial-based reorganization on the trajectories with the following steps. First, we divide the road network into grids, and then split individual trajectory into shorter trajectories based on these grids. Next, we put the trajectories within the same gird into the same file. Finally, we do the compression on reorganized files. Due to the reorganization, similar trajectories are more likely to fall into the same sliding window of LZMA. As a result, the compression ratio is relatively good even when the partition size of the compression algorithm is quite small, e.g., 32KB. Moreover, smaller partition size usually means less memory consumption. Hence, one of the benefits of the reorganization is that

it reduces the amount of memory and processing time required to achieve same compression ratios.

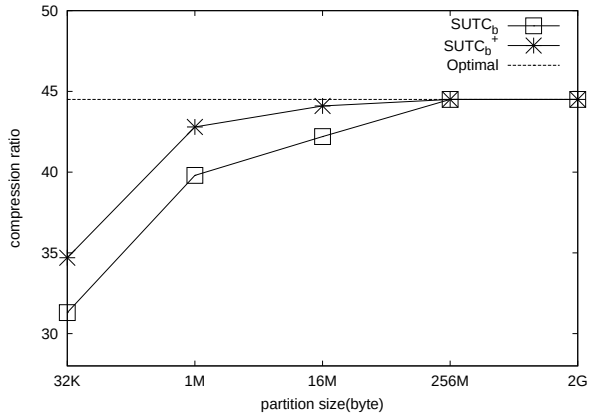

Figure 8. Effects of Trajectory Reorganization ($N = 10,000, \Delta t = 15s, \delta = 30m$. The SUTC$_b$ algorithm with reorganization is referred to as SUTC$_b^+$ for short).

Table 1 shows the time required by these algorithms to process about 25 million trajectory points. As indicated in Table 1, SUTC$_b$ is quite competitive to DP$_z$, and both of them are more efficient than the simple LZMA. Compared to SUTC$_b$, SUTC$_v$ is much slower. This is mainly because SUTC$_v$ involves the map-matching and a lot of time-consuming shortest path calculation while computing the velocity between two adjacent trajectory points.

Table 1. Efficiency (The number of trajectories $N = 10,000$ which contains 25+ million trajectory points, $\Delta t = 15s, \delta = 30m$; MapM: Map Matching, TSym: Trajectory Symbolization, LZMA:The Lempel-Ziv-Markov chain algorithm in 7Zip).

Algs	Step 1: MapM	Step 2: TSym	Step 3: LZMA	Total (sec.)
LZMA	0	0	813	813
DP$_z$	0	114	221	335
SUTC$_b$	0	169	331	500
SUTC$_v$	3883	376	314	4573

5. Conclusion

In this paper, we investigate the problem of how to compress large scale urban trajectory data, and propose a scalable urban trajectory compression solution called SUTC. The experimental results on real dataset demonstrate that SUTC algorithms manage to achieve much higher compression ratios than the representative DP compression algorithm when processing large scale trajectory data. In addition, the efficiency of SUTC is quite competitive to DP. Therefore, compared to existing trajectory compression algorithms, the proposed algorithms are more suitable for processing large scale urban trajectory data. In the future work, we will improve SUTC algorithms to better handle low-sampling-rate trajectories and investigate the compression engine with indexing components to support traditional queries on compressed data.

Acknowledgments

We would like to thank Prof. Xiaofang Zhou for useful discussions. We also acknowledge the help from Dr. Jiajie Xu and Dr. Xike Xie for providing valuable suggestions.

This work was supported by the National Natural Science Foundation of China (No. 61202064), the National High Technology Research and Development Program of China (863 Program) (No. 2013AA01A603), and the Strategic Priority Research Program of the Chinese Academy of Sciences (No. XDA06010600).

References

[1] E. G. Agency. Global satellite navigation system (gnss) market report. Technical report, Oct 2013.

[2] Y. Cai and R. Ng. Indexing spatio-temporal trajectories with chebyshev polynomials. In *SIGMOD*, pages 599–610, 2004.

[3] W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *Int. J. of Comput. Geom. Appl.(IJCGA)*, 6(1):59–77, 1996.

[4] A. Civilis, C. S. Jensen, and S. Pakalnis. Techniques for efficient road-network-based tracking of moving objects. *IEEE Trans. on Knowl. and Data Eng.*, 17(5):698–712, May 2005.

[5] 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. *Canadian Cartographer*, 10:112–122, 1973.

[6] R. Gotsman and Y. Kanza. Compact representation of gps trajectories over vectorial road networks. In *SSTD*, pages 241–258, 2013.

[7] J. Gudmundsson, J. Katajainen, D. Merrick, C. Ong, and T. Wolle. Compressing spatio-temporal trajectories. *Computational Geometry – Theory and Applications*, online, Feb. 2009.

[8] N. Hönle, M. Grossmann, S. Reimann, and B. Mitschang. Usability analysis of compression algorithms for position data streams. In *GIS*, pages 240–249, 2010.

[9] G. Kellaris, N. Pelekis, and Y. Theodoridis. Trajectory compression under network constraints. In *SSTD*, pages 392–398, 2009.

[10] M. Koegel, W. Kiess, M. Kerper, and M. Mauve. Compact Vehicular Trajectory Encoding. In *VTC*, pages 1–5, 2011.

[11] R. Lange, F. Dürr, and K. Rothermel. Efficient real-time trajectory tracking. *The VLDB Journal*, 20(5):671–694, Oct. 2011.

[12] Y. Li, C. Liu, K. Liu, J. Xu, F. He, and Z. Ding. On efficient map-matching according to intersections you pass by. In *DEXA*, pages 42–56, 2013.

[13] B. Lin and J. Su. One way distance: For shape based similarity search of moving object trajectories. *Geoinformatica*, 12(2):117–142, 2008.

[14] K. Liu, K. Deng, Z. Ding, X. Zhou, and M. Li. Pattern-based moving object tracking. In *SIGSPATIAL TDMA*, pages 5–14, 2011.

[15] K. Liu, Y. Li, F. He, J. Xu, and Z. Ding. Effective map-matching on the most simplified road network. In *GIS*, pages 609–612, 2012.

[16] N. Meratnia and R. de By. Spatiotemporal compression techniques for moving point objects. In *EDBT*, pages 765–782, 2004.

[17] J. Muckell, J.-H. Hwang, C. T. Lawson, and S. S. Ravi. Algorithms for compressing gps trajectory data: an empirical evaluation. In *GIS*, pages 402–405, 2010.

[18] M. Potamias, K. Patroumpas, and T. Sellis. Sampling trajectory streams with spatiotemporal criteria. In *SSDBM*, pages 275–284, 2006.

[19] D. Sacharidis, K. Patroumpas, M. Terrovitis, V. Kantere, M. Potamias, K. Mouratidis, and T. Sellis. On-line discovery of hot motion paths. In *EDBT*, pages 392–403, 2008.

[20] F. Schmid, K.-F. Richter, and P. Laube. Semantic trajectory compression. In *SSTD*, pages 411–416, 2009.

[21] G. Trajcevski, H. Cao, P. Scheuermann, O. Wolfson, and D. Vaccaro. On-line data reduction and the quality of history in moving objects databases. In *MobiDE*, pages 19–26, 2006.

[22] T. Westmann, D. Kossmann, S. Helmer, and G. Moerkotte. The implementation and performance of compressed databases. *SIGMOD Rec.*, 29:55–67, Sep 2000.

[23] Y. Zheng and X. Zhou, editors. *Computing with Spatial Trajectories*. Springer, 2011.