An Online Approach for Direction-Based Trajectory Compression with Error Bound Guarantee

Bingqing Ke, Jie Shao(✉), Yi Zhang, Dongxiang Zhang, and Yang Yang

School of Computer Science and Engineering,
University of Electronic Science and Technology of China, Chengdu, China
{201521060331,yizhang}@std.uestc.edu.cn,
{shaojie,zhangdo,yang.yang}@uestc.edu.cn

Abstract. With the increasing usage of GPS-enabled devices which can record users' travel experiences, moving object trajectories are collected in many applications. Raw trajectory data can be of large volume but storage is limited, and direction-based compression to preserve the skeleton of a trajectory became popular recently. In addition, real-time applications and constrained resources often require online processing of incoming data instantaneously. To address this challenge, in this paper we first investigate two approaches extended from Douglas-Peucker and Greedy Deviation algorithms respectively, which are two most popular algorithms for trajectory compression. To further improve the online computational efficiency, we propose a faster approximate algorithm with error bound guarantee named *Angular-Deviation*. Experimental results demonstrate it can achieve low running time to suit the most constrained computation environments.

Keywords: GPS trajectory · Direction-based compression · Online algorithm

1 Introduction

With the popularity of GPS-enabled devices, trajectory data which records the traces of moving objects by sampling their positions according to a certain sampling rate is becoming ubiquitous. The raw trajectory data is massive, which leads to expensive storage and processing cost. Trajectory clustering employs various clustering algorithms (e.g., [13]), and representative trajectories can be extracted for dataset profiling [3]. Compressing each raw trajectory with a subset of important points is more commonly used, by eliminating points that contain little information. Moreover, the emergence of real-time applications and constrained resources demand intelligent online algorithms that can process the incoming points instantaneously.

There are many methods in the literature for trajectory compression [1,2,9,11,12] and most of them aim at preserving the position information. Existing online

F. Li et al. (Eds.): APWeb 2016, Part I, LNCS 9931, pp. 79–91, 2016.
DOI: 10.1007/978-3-319-45814-4_7

compression algorithms [5,6,10] also make the traditional assumption that the goal should be to compress trajectories such that the position information captured in the compressed trajectories is "similar" to the position information captured in the raw trajectories. We call them *position-based online trajectory compression* algorithms.

Recently, an alternative type of direction-based trajectory compression which aims at preserving direction information is proposed in [7], and demonstrated its superiority compared with position-based methods. In this paper, in order to solve this new type of trajectory compression in an online fashion, we first investigate two baseline solutions, namely *Direction-based Buffered Douglas-Peucker* (DBDP) and *Direction-based Buffered Greedy Deviation* (DBGD), by extending Douglas-Peucker [2] and Greedy Deviation (a variation of the generic sliding window algorithm [4]) respectively. The time complexities of DBDP and DBGD are both $\mathcal{O}(Bn)$, where B represents the size of buffer and n represents the number of trajectory points. To suit the more constrained computation environments, we further propose a faster approximate algorithm with error bound guarantee named *Angular-Deviation*, which can achieve $\mathcal{O}(n)$ time complexity.

In summary, we make the following contributions in this paper:

- We study online processing of direction-based trajectory compression, which is a novel problem and has important applications in practice.
- We introduce two baseline algorithms for online trajectory compression with $\mathcal{O}(Bn)$ time complexity, where B denotes the buffer size. To further improve the efficiency of online compression, a faster algorithm named *Angular-Deviation* which achieves $\mathcal{O}(n)$ time complexity is given.
- We conduct extensive experiments to verify that the proposed *Angular-Deviation* algorithm achieves lower running time than two baseline algorithms.

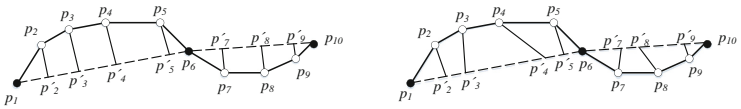
The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 introduce the property of direction-based trajectory compression. We present two baseline solutions in Sect. 4, and then design a faster version of online algorithm in Sect. 5. We show the experimental results in Sect. 6. Finally, we conclude in Sect. 7.

2 Related Work

2.1 Error Measurements for Trajectory Compression

According to different error measurements adopted, we can divide existing work on trajectory compression into two categories: (1) position-based [1,2,5,6,9–12]; and (2) direction-based [7,8].

Position-Based Trajectory Compression. A position-based error of a compressed trajectory T' denoted by $\varepsilon_p(T')$ is usually defined to be the maximum Euclidean distance between a position on the original trajectory and its "mapped" position p' on the compressed trajectory. There are mainly two distance metrics to measure the position-based error of a compression: *perpendicular*



(a) Perpendicular Euclidean Distance (b) Time Synchronized Euclidean Distance

Fig. 1. Distance metrics to measure the position-based compression error.

Euclidean distance and *time synchronized Euclidean distance* [12]. As illustrated in Fig. 1, suppose we compress a trajectory with 10 points into a representation T' with 3 points. The former metric defines the "mapped" position to be the closest position from p on the compressed trajectory and $\varepsilon_p(T') = dis(p_3, p'_3)$. The latter metric assumes that the "mapped" position is the position with the same timestamp on the compressed trajectory as p, and we get $\varepsilon_p(T') = dis(p_4, p'_4)$.

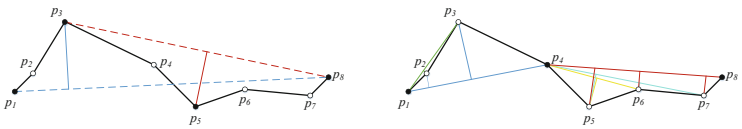
Direction-Based Trajectory Compression. Direction-preserving compression [7] is the state-of-the-art method for trajectory compression. The error of the compressed trajectory is the maximum angular difference between a segment and its corresponding compressed segment. In this work, we focus on direction-based trajectory compression for the reason of its superiority (Lemma 1). The details will be elaborated in Sect. 3.

2.2 Existing Trajectory Compression Methods

In this part, we review two representative trajectory compression methods.

Douglas-Peucker. As demonstrated in Fig. 2(a), the idea of Douglas-Pecker algorithm [1] is to replace the original trajectory by an approximate line segment (e.g., $\overline{p_1 p_8}$). If the replacement does not meet the specified error requirement (perpendicular Euclidean distance is used here), it recursively partitions the original trajectory into two sub-trajectories (e.g., $\{p_1,p_2,p_3\}$ and $\{p_3,p_4,p_5,p_6,p_7,p_8\}$) by selecting the point contributing the largest error as the splitting point (e.g., p_3).

Sliding Window. The major limitation of the above Douglas-Peucker algorithm is that it runs offline. This is not sufficient for location-aware applications which often require real-time data processing. A generic sliding window algorithm can run online. The idea of the sliding window algorithm [4] is to append the points



(a) Douglas-Peucker (b) Sliding Window

Fig. 2. Existing trajectory compression methods.

to a growing sliding window and continue to grow the sliding window until the approximation error exceeds some error tolerance. As illustrated in Fig. 2(b), p_4 will be first reserved as the error for p_3 exceeds the threshold. Then, the algorithm starts from p_5 and reserves p_8. Other points are discarded.

Both of the two methods aim at preserving position information, and cannot be directly used for direction-based trajectory compression. We develop two baseline solutions extended from these two methods in Sect. 4.

3 Preliminaries

First, we give some definitions about trajectory compression.

Definition 1 (Trajectory). *A trajectory is represented by a sequence of n points in the form of* $\{p_1(x_1, y_1, t_1), p_2(x_2, y_2, t_2), ..., p_n(x_n, y_n, t_n)\}$, *where* (x_i, y_i) *denotes the longitude and latitude information of point* p_i *at timestamp* t_i.

Definition 2 (Compressed Trajectory). *Given a trajectory that contains n points* $T = \{p_1, p_2, ..., p_n\}$, *its compressed trajectory is* $T' = \{p_{s_1}, p_{s_2}, ..., p_{s_m}\}$ *where* $s_1 = 1$, $s_m = n$ *and* $T' \subseteq T$.

The notion of direction-based error for compression [7] is used in our work.

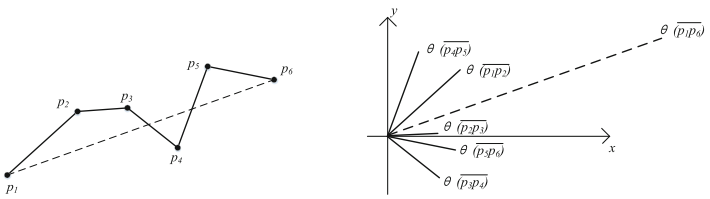
Definition 3 (Direction-Based Error of Compression). *Let the maximum value of angular difference between segment* $\overline{p_i p_{i+1}}$ *and corresponding simplified segment* $\overline{p_{s_k} p_{s_{k+1}}}$ *be* $\varepsilon(\overline{p_{s_k} p_{s_{k+1}}})$. *As shown in Fig. 3,* $\overline{p_{s_k} p_{s_{k+1}}}$ *is a simplified segment, and each segment* $\overline{p_i p_{i+1}}$ *between* p_{s_k} *and* $p_{s_{k+1}}$ *has an angular difference* $\theta(\overline{p_i p_{i+1}}, \overline{p_{s_k} p_{s_{k+1}}})$. *We define* $\varepsilon(\overline{p_{s_k} p_{s_{k+1}}})$ *as:*

$$\varepsilon(\overline{p_{s_k} p_{s_{k+1}}}) = \max_{p_{s_k} \leq i < p_{s_{k+1}}} |\theta(\overline{p_i p_{i+1}}) - \theta(\overline{p_{s_k} p_{s_{k+1}}})|.$$

Now, we can define the error of the whole compression as:

$$\varepsilon(T') = \max_{1 \leq k < m} \{\varepsilon(\overline{p_{s_k} p_{s_{k+1}}})\},$$

where T' *is the result of the compression, and* m *is the number of points that* T' *contains. We regard* $(n - m)/n$ *as the compression rate when the raw trajectory is composed of* n *points.*



(a) Trajectory (b) Direction

Fig. 3. Direction-based error measurement.

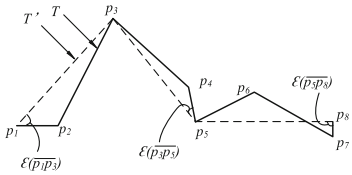


Fig. 4. An example illustrating direction-based error measurement.

Example 1. As shown in Fig. 4, a trajectory T is in the form of $\{p_1, p_2, ..., p_8\}$, and the compressed trajectory T' is $\{p_1, p_3, p_5, p_8\}$. Consider our example, $\varepsilon(\overline{p_1p_3}) = \pi/4$, $\varepsilon(\overline{p_3p_5}) = \pi/6$ and $\varepsilon(\overline{p_5p_8}) = \pi/2$. Thus, $\varepsilon(T') = \max\{\varepsilon(\overline{p_1p_3}), \varepsilon(\overline{p_3p_5}), \varepsilon(\overline{p_5p_8})\} = \varepsilon(\overline{p_5p_8}) = \pi/2$.

Direction-based trajectory compression not only preserves the direction information but also bounds position information loss. In this following, we give a brief proof.

Lemma 1 (Bounded Error of Distance). *The distance error of direction-based trajectory compression d_e meets the inequation*

$$d_e \leq \frac{1}{2} L_{max} \tan \varepsilon$$

where L_{max} is the maximum length of all segments in the compressed trajectory, and ε is the error tolerance of the compression.

Proof. An illustration is shown in Fig. 5, where $\overline{p_{s_k}p_{s_{k+1}}}$ is a compressed segment. We first prove that any p_i between p_{s_k} and $p_{s_{k+1}}$ in original trajectory is inside the rhomboid $\diamond_{ap_{s_k}bp_{s_{k+1}}}$. Assume that p_i is outside the rhomboid $\diamond_{ap_{s_k}bp_{s_{k+1}}}$ and it is over the segment $\overline{p_{s_k}a}$ as shown in Fig. 5, there must exist such a segment $\overline{p_{j-1}p_j}$ that intersects the segment $\overline{p_{s_k}a}$. Then the angular difference between $\overline{p_{j-1}p_j}$ and $\overline{p_{s_k}p_{s_{k+1}}}$ must exceed ε. This contradicts with the definition of the direction-based error of compression (Definition 4), and thus p_i must be inside the rhomboid $\diamond_{ap_{s_k}bp_{s_{k+1}}}$. Next, if p is the furthest point from $\overline{p_{s_k}p_{s_{k+1}}}$, we have

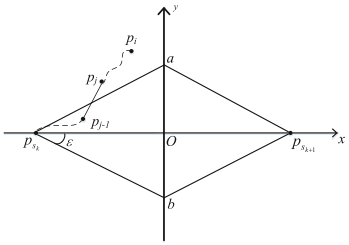


Fig. 5. Illustration of Lemma 1.

$$\begin{aligned} dis(p, \overline{p_{s_k} p_{s_{k+1}}}) &\leq dis(a, \overline{p_{s_k} p_{s_{k+1}}}) = dis(a, O) \\ &= \frac{1}{2} dis(p_{s_k}, p_{s_{k+1}}) \tan \varepsilon \leq \frac{1}{2} L_{max} \tan \varepsilon. \end{aligned} \tag{1}$$

4 Baseline Solutions

In this section, we introduce two baseline solutions for *direction-based online trajectory compression*, namely *Direction-based Buffered Douglas-Peucker* (DBDP) and *Direction-based Buffered Greedy Deviation* (DBGD). They are extended from Douglas-Peucker and sliding window methods, respectively.

4.1 Direction-Based Buffered Douglas-Peucker

The major idea of Direction-based Douglas-Peucker (DDP) is to recursively split the trajectory into sub-trajectories, at the point which is the end of the segment that has the largest angular difference from the approximated segment (linking the start point and the end point of this trajectory), and finally we approximate each sub-trajectory until there are $W-1$ sub-trajectories where W is the number of points the resulting trajectory contains.

The DDP algorithm still runs offline, and its time complexity is $\mathcal{O}(n^2)$, which is inapplicable for data streams. To solve the problem, Direction-based Buffered Douglas-Peucker (DBDP) can be used. In this strategy, the incoming points are accumulated in a buffer, and then the points are processed by applying DDP algorithm when the buffer is full. Such a solution has an inferior compression rate, because at least two points should be kept whenever the buffer is full.

Complexity Analysis. Suppose a trajectory contains n points and the buffer size is B. Since there are $\mathcal{O}(n/B)$ times of calling DDP algorithm which causes $\mathcal{O}(B^2)$ time complexity in the worst case, DBDP has a time complexity of $\mathcal{O}(B^2 \cdot (n/B)) = \mathcal{O}(Bn)$.

4.2 Direction-Based Buffered Greedy Deviation

Direction-based Buffered Greedy Deviation (DBGD) is a variation of the generic sliding window algorithm. In this strategy, if the buffer is not full, whenever a point p_e arrives we append the point to the end of the buffer, and calculate the error of the compressed trajectory segment defined by the start point in the buffer and the end point p_e. If the error already exceeds the tolerance, we reverse the last point, clear the buffer and start a new compression at the last point. Otherwise, the process waits for the next incoming point. It is easy to find that the algorithm guarantees the error tolerance. However, its major weakness is the relatively high time complexity, since a complete calculation of the error is needed as long as a new point arrives, which is undesirable in online processing.

Complexity Analysis. For the i^{th} incoming point in the buffer, we should calculate errors for $\mathcal{O}(i)$ times. In the worst case, for all B points in the buffer, calculations have to be made $\mathcal{O}(1 + 2 + 3 + ... + B) = \mathcal{O}(B^2)$ times. Therefore, the time complexity of DBGD is $\mathcal{O}(B^2 \cdot (n/B)) = \mathcal{O}(Bn)$.

5 Approximate Solution with Error Bound Guarantee

As discussed in Sect. 4, the complexities of two baseline solutions may not satisfy stringent performance requirements in resource-constrained environments. To improve the computational efficiency, we propose an approximate Angular-Deviation algorithm with guaranteed error bounds.

5.1 Definitions

Definition 4 (Angular Deviation). *The angular deviation $p_i.\varepsilon_d$ is the difference between two consecutive segments' directions,*

$$\Delta\theta = \theta(\overline{p_i p_{i+1}}) - \theta(\overline{p_{i-1} p_i}) \tag{2}$$

$$p_i.\varepsilon_d = \begin{cases} \Delta\theta + 2\pi & \Delta\theta \leq -\pi \\ \Delta\theta & -\pi < \Delta\theta \leq \pi \\ \Delta\theta - 2\pi & \Delta\theta > \pi \end{cases} \tag{3}$$

where $\theta(\overline{p_i p_{i+1}})$ represents the direction of the segment $\overline{p_i p_{i+1}}$, namely the angle between vector $\overrightarrow{p_i p_{i+1}}$ and the positive direction of x-axis. We set the range of $\theta(\overline{p_i p_{i+1}})$ as $(-\pi, \pi]$. In Fig. 6, $p_2.\varepsilon_d$ and $p_3.\varepsilon_d$ represent the change of direction of p_2 and p_3 respectively. Note that, the value of the angular deviation has a positive or negative sign.

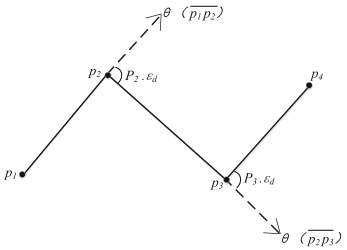


Fig. 6. Angular deviation ε_d of p_i.

Definition 5 (Accumulated Angular Deviation). *The accumulated angular deviation ε_a of point p_i is defined as:*

$$p_i.\varepsilon_a = \begin{cases} 0 & i = s_k \\ \sum_{m=s_k+1}^{i} p_m.\varepsilon_d & s_k < i < s_{k+1} \end{cases} \tag{4}$$

where p_{s_k} is the start point of the compressed segment $\overline{p_{s_k} p_{s_{k+1}}}$.

Combining with the Definition 4, we have another form of the expression of $p_i.\varepsilon_a$:

$$\Delta\phi = \theta(\overline{p_i p_{i+1}}) - \theta(\overline{p_{s_k} p_{s_k+1}}) \tag{5}$$

$$p_i.\varepsilon_a = \begin{cases} \Delta\phi + 2\pi & \Delta\phi \leq -\pi \\ \Delta\phi & -\pi < \Delta\phi \leq \pi \\ \Delta\phi - 2\pi & \Delta\phi > \pi \end{cases} \tag{6}$$

where $s_k \leq i < s_{k+1}$.

5.2 Theoretical Property

Before introducing the main idea of the approximate algorithm, we first introduce a fundamental lemma.

Lemma 2 (Bounded Error of Direction). *The error of the compression* $\varepsilon(T')$ *meets the inequation*

$$\varepsilon(T') < 2\varepsilon_t$$

where ε_t *is the threshold of accumulated angular deviation.*

Proof. As we defined before, the direction of any segment $\overline{p_i p_{i+1}}$ between p_{s_k} and $p_{s_{k+1}}$ can be calculated by Eqs. 4, 5 and 6:

$$\Psi = \theta(\overline{p_{s_k} p_{s_k+1}}) + \sum\nolimits_{m=s_k+1}^{i} p_m.\varepsilon_d$$

$$\theta(\overline{p_i p_{i+1}}) = \begin{cases} \psi + 2\pi & \psi \leq -\pi \\ \psi & -\pi < \psi \leq \pi \\ \psi - 2\pi & \psi > \pi \end{cases}$$

As shown in Fig. 7, the direction of any segment $\theta(\overline{p_i p_{i+1}})$ between p_{s_k} and $p_{s_{k+1}}$ are limited within interval $[\theta(\overline{p_{s_k} p_{s_k+1}}) - \varepsilon_t, \theta(\overline{p_{s_k} p_{s_k+1}}) + \varepsilon_t]$. It is easy to find that the direction of compressed segment $\theta(\overline{p_{s_k} p_{s_{k+1}}})$ meets the inequation:

$$\theta(\overline{p_{s_k} p_{s_k+1}}) - \varepsilon_t < \theta(\overline{p_{s_k} p_{s_{k+1}}}) < \theta(\overline{p_{s_k} p_{s_k+1}}) + \varepsilon_t.$$

Therefore, the maximum angular difference between any segment and the simplified segment is less than $2\varepsilon_t$.

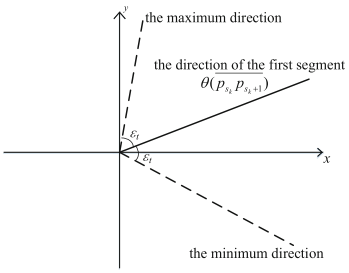


Fig. 7. Illustration of Lemma 2.

5.3 Angular-Deviation Algorithm

The *Angular-Deviation* algorithm is formally described in Algorithm 1. Besides, we present an example to show the details of the process.

Algorithm 1. Angular-Deviation for Direction-based Online Compression

Initialization:

Set the accumulated change of direction $\varepsilon_a = 0$, the resulting set $Traj = \emptyset$;

Input:

Start point s, the point currently being processed p_i, the previous point p_{i-1}, the new incoming point p_{i+1}, an error tolerance $error$, the threshold of accumulated angular deviation $\varepsilon_t = error/2$;

1: **if** p_{i+1}=null **then**
2: $Traj \leftarrow Traj \bigcup \{p_i\}$;
3: **return** $Traj$; $//p_i$ is the end of the whole trajectory
4: **else**
5: $p_i.\varepsilon_d \leftarrow AngDiff(p_{i-1}, p_i, p_{i+1})$;
6: $\varepsilon_a = \varepsilon_a + p.\varepsilon_d$;
7: **if** $|\varepsilon_a| > \varepsilon_t$ **then**
8: $Traj \leftarrow Traj \bigcup \{p_i\}$;
9: $\varepsilon_a \leftarrow 0$;
10: **end if**
11: **end if**

As shown in Algorithm 1, we set two variables: ε_a is the accumulated change of direction of the incoming point p_i, and $Traj$ is the resulting set which contains the compressed trajectory we expect (initialized as empty). Then we set two tolerance values: $error$ denotes the error tolerance of the compression and ε_t is equal to half of $error$ according to Lemma 2. The algorithm firstly checks if the current point p_i is the end point of the trajectory (line 1), because the end point must be reserved (line 2). If p_i is an intermediate point, the accumulated angular deviation ε_a is calculated by Eq. 4 (lines 5–6). If $|\varepsilon_a| > \varepsilon_t$, we cannot guarantee that the maximum angular difference between a segment on original trajectory and corresponding compressed segment less than $error$ by Lemma 2, so we should reverse p_i and start a new segment compression (lines 8–9), otherwise p_i should be discarded since it carries little direction information.

Example 2. Consider an example shown in Fig. 8, $T = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$, and we set $error = \pi/2 = 1.57$ radian and $\varepsilon_t = \pi/4 = 0.785$ radian. Firstly, $p_i = p_2$ and $|\varepsilon_a| = |p_2.\varepsilon_d = +0.785| \leq \varepsilon_t$, hence p_2 should be deleted. Then, $p_3.\varepsilon_d = -2.356$ and we can get $|\varepsilon_a| = |0.785 - 2.356| = 1.571 > \varepsilon_t$, hence $Traj = \{p_1, p_3\}$ and $\varepsilon_a = 0$. Then, the algorithm starts from p_4, $|\varepsilon_a| = p_4.\varepsilon_d = +0.524 < \varepsilon_t$. Next, $p_5.\varepsilon_d = +1.571$, the accumulated angular deviation $|\varepsilon_a| = 0.524 + 1.571 = 2.095 > \varepsilon_t$, hence $Traj = \{p_1, p_3, p_5\}$. After that, p_6 is deleted and p_7 is reversed in the same way. Finally, we get the compressed trajectory $Traj = \{p_1, p_3, p_5, p_7\}$.

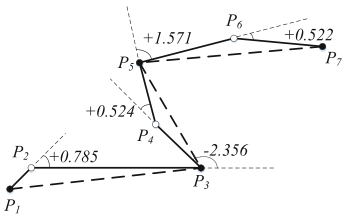
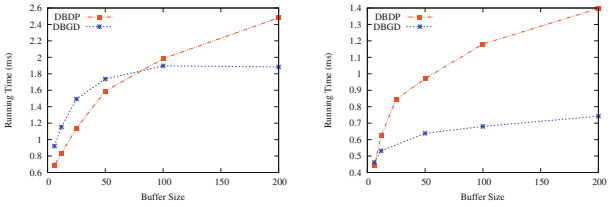


Fig. 8. An example illustrating Angular-Deviation algorithm.

6 Experiments

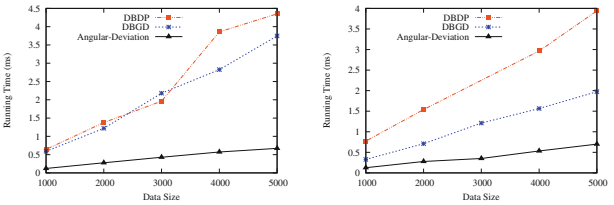
We conducted extensive experiments to evaluate two baseline algorithms DBDP and DBGD, and approximate algorithm *Angular-Deviation*. All algorithms are implemented in C++ and run on a Linux platform (Intel Xeon E5-2620 2.1GHz CPU with 6 cores and 200GB RAM). Two real datasets are used: (1) Chengdu taxi dataset, which contains taxi trajectories in Chengdu, China in August 2014 and each trajectory contains 3038 points on average; (2) T-Drive dataset which contains taxi trajectories generated by over 10000 taxis in a period of one week in Beijing [14].

Running Time with Buffer Size: In this experiment, as *Angular-Deviation* algorithm does not require a buffer, we study the effect of buffer size (i.e., B) for two baseline algorithms DBDP and DBGD. The values used for buffer size B are 6, 12, 25, 50, 100, 200 (the error tolerance ε is fixed to be 1). The running time results are shown in Fig. 9. We can see that the running time of both DBDP and DBGD will rise linearly with the increase of B. This is reasonable as the time complexities of both algorithms are $\mathcal{O}(Bn)$. Besides, it is worth noting that the running time of DBGD becomes stable when the buffer size is larger than some value. This is because that if buffer is large enough, the error of the compressed trajectory segment will always exceed ε before the buffer is full, which leads to the situation that buffer size has little impact on the compression. Therefore, the running time no longer depends on buffer size.



(a) Chengdu Taxi Dataset (b) T-Drive Dataset

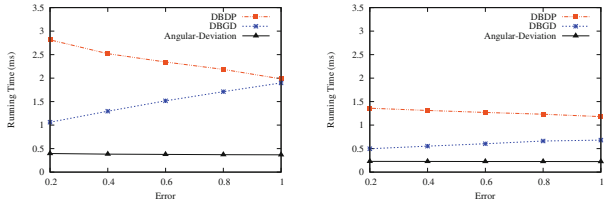
Fig. 9. Effect of buffer size B.



(a) Chengdu Taxi Dataset (b) T-Drive Dataset

Fig. 10. Effect of data size $|T|$.

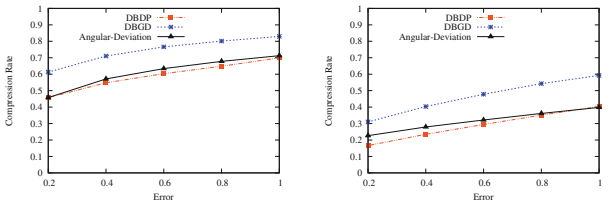
Running Time with Data Size: In this experiment, we study the effect of data size (i.e., $|T|$) on the performance of three algorithms (DBDP, DBGD and *Angular-Deviation*). The values used for $|T|$ are around 1 k, 2 k, 3 k, 4 k and 5 k (error tolerance ε is fixed to be 1 and buffer size B is fixed to be 100). For each data size, we select a set of 10 trajectories each of which has its size close to this value and run three algorithms on each of these trajectories. Then, we average the experimental results on these trajectories. The result is shown in Fig. 10. As we can see, *Angular-Deviation* runs much faster than the other two algorithms due to its low time complexity. Besides, we observe that the running time of all three algorithms increases approximatively linearly with data size. This is because the time complexity of each algorithm is proportional to data size (DBDP and DBGD are $\mathcal{O}(Bn)$, and *Angular-Deviation* is $\mathcal{O}(n)$).



(a) Chengdu Taxi Dataset (b) T-Drive Dataset

Fig. 11. Effect of error tolerance ε.

Running Time with Error Tolerance: We increase the error tolerance ε from 0.2 to 1 and report the results in Fig. 11 (buffer size B is set to be 100). As shown in Fig. 11, *Angular-Deviation* achieves the best performance, and the running time is not sensitive to ε. This is because whatever ε is, *Angular-Deviation* just needs to calculate the direction of each segment and the angular difference. The running time of DBDP decreases when ε becomes larger. The reason is that the number of segments which leads to a split (Sect. 4.1) is reduced. For DBGD, when ε increases, it needs more time to compress which is contrast to DBDP. Consider that we give each point a number in ascending order when it is added



(a) Chengdu Taxi Dataset (b) T-Drive Dataset

Fig. 12. Compression rate.

in the buffer. For i^{th} point in the buffer, it leads to $i + 1$ times of calculation (2 times for direction and $i - 1$ times for angular difference). That is, the larger number the point has, the more calculations caused. Hence, when ε becomes larger, the number of points that have relatively large number decreases, and the running time is reduced consequently.

Compression Rate with Error Tolerance: In this experiment, we compare the compression rate with increasing error tolerance ε and the experimental results are shown in Fig. 12. The compression rate β is defined as $\beta = (|T| - |T'|)/|T|$, where T denotes the number of raw trajectory and T' represents the result of compression. For β, higher is better. When error tolerance ε becomes larger, compression rates of three algorithm all increase. That is because the number of segments which cause the error of compression exceeds ε decreases. The DBDP algorithm achieves the worst performance. Since for every B points in the buffer, besides the points that cause unacceptable errors there are at least two points to be reversed. The *Angular-Deviation* algorithm performs worse than DBGD. The reason is that, for the incoming point, DBGD deletes the points that exactly cause that the error of compression exceeds threshold. Unlike DBGD, *Angular-Deviation* does not require to calculate the exact error of compression, and it discards the points with accumulated angular deviation larger than $1/2\varepsilon$, which just probably leads to an error that exceeds ε by Lemma 2.

7 Conclusion and Future Work

This paper studies a novel problem of online algorithms for direction-based trajectory compression. We first introduce two baseline solutions extended from Douglas-Peucker and Greedy Deviation algorithms respectively. To further improve the computational efficiency, we propose a faster *Angular-Deviation* algorithm with error bound guarantee. Experimental results show that *Angular-Deviation* runs fastest. All methods on trajectory compression mentioned in this paper aim at preserving skeleton information of trajectory in Euclidean space. In the future, we plan to study how to compress trajectories on road networks.

Acknowledgments. This work is support in part by the Fundamental Research Funds for the Central Universities (No. ZYGX2015J058, No. ZYGX2015J055 and No. ZYGX2014Z007), and the National Nature Science Foundation of China (No. 61572108).

References

1. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Can. Cartographer **10**(2), 112–122 (1973)
2. Heckbert, P.S., Garland, M.: Survey of polygonal surface simplification algorithms. Technical report, Carnegie Mellon University (1997)
3. Jiang, W., Zhu, J., Xu, J., Li, Z., Zhao, P., Zhao, L.: HV: a feature based method for trajectory dataset profiling. In: Cellary, W., Wang, D., Wang, H., Chen, S.-C., Li, T., Zhang, Y. (eds.) WISE 2015. LNCS, vol. 9418, pp. 46–60. Springer, Heidelberg (2015). doi:10.1007/978-3-319-26190-4_4
4. Keogh, E.J., Chu, S., Hart, D.M., Pazzani, M.J.: An online algorithm for segmenting time series. In: ICDM, pp. 289–296 (2001)
5. Kolesnikov, A.: Efficient online algorithms for the polygonal approximation of trajectory data. In: MDM, pp. 49–57 (2011)
6. Liu, J., Zhao, K., Sommer, P., Shang, S., Kusy, B., Jurdak, R.: Bounded quadrant system: error-bounded trajectory compression on the go. In: ICDE, pp. 987–998 (2015)
7. Long, C., Wong, R.C., Jagadish, H.V.: Direction-preserving trajectory simplification. PVLDB **6**(10), 949–960 (2013)
8. Long, C., Wong, R.C., Jagadish, H.V.: Trajectory simplification: on minimizing the direction-based error. PVLDB **8**(1), 49–60 (2014)
9. Meratnia, N., Park, Y.-Y.: Spatiotemporal compression techniques for moving point objects. In: Bertino, E., Christodoulakis, S., Plexousakis, D., Christophides, V., Koubarakis, M., Böhm, K. (eds.) EDBT 2004. LNCS, vol. 2992, pp. 765–782. Springer, Heidelberg (2004)
10. Muckell, J., Hwang, J., Patil, V., Lawson, C.T., Ping, F., Ravi, S.S.: SQUISH: an online approach for GPS trajectory compression. In: COM.Geo, pp. 13:1–13:8 (2011)
11. Muckell, J., Olsen, P.W., Hwang, J., Lawson, C.T., Ravi, S.S.: Compression of trajectory data: a comprehensive evaluation and new approach. GeoInformatica **18**(3), 435–460 (2014)
12. Potamias, M., Patroumpas, K., Sellis, T.K.: Sampling trajectory streams with spatiotemporal criteria. In: SSDBM, pp. 275–284 (2006)
13. Yang, Y., Ma, Z., Yang, Y., Nie, F., Shen, H.T.: Multitask spectral clustering by exploring intertask correlation. IEEE Trans. Cybern. **45**(5), 1069–1080 (2015)
14. Yuan, J., Zheng, Y., Zhang, C., Xie, W., Xie, X., Sun, G., Huang, Y.: T-drive: driving directions based on taxi trajectories. In: ACM-GIS, pp. 99–108 (2010)