2017 IEEE 18th International Conference on Mobile Data Management

An Efficient Online Approach for Direction-Preserving Trajectory Simplification with Interval Bounds

Bingqing Ke Jie Shao Dongxiang Zhang
School of Computer Science and Engineering
University of Electronic Science and Technology of China
kebingqing@std.uestc.edu.cn {shaojie,zhangdo}@uestc.edu.cn

***Abstract*—The prevalence of GPS devices has facilitated collection of large-scale trajectories. Fresh positions of moving objects can be sampled periodically and sent to servers for data analytics and query processing. Online trajectory simplification is a compression task usually conducted at the sensor side and serves as a key component to reduce network communication overhead. In this paper, we study a new trajectory simplification problem which is direction-preserving and works in an online fashion. An efficient simplification algorithm is proposed, which is guaranteed to be error-bounded and achieves $\mathcal{O}(n)$ time and $\mathcal{O}(1)$ space complexity. In an extensive experimental evaluation with two real datasets, our approach exhibits superior performance on both running time and compression rate.**

***Keywords*-GPS trajectory; direction-preserving simplification; online algorithm**

I. Introduction

The prevalence of GPS devices has drastically boosted the scale of trajectory collection. The volume of raw trajectory data collected from moving objects is often too large to be stored and processed. For instance, given a city with 3 millions vehicles sampled every five seconds, the size of the collected trajectories for just one day can reach nearly 1 TB (suppose 12 bytes are required for the latitude, longitude and timestamp). One of the standard techniques to reduce the storage cost is called trajectory simplification. Its main idea is to drop certain samples on purpose but guarantee an error bound between the simplified and original trajectory. Such simplification process, when conducted at the sensor side, can save considerable amounts of communication cost to transfer the sampled GPS data. In addition, the emergence of real-time applications such as real-time traffic monitoring and real-time route recommendation require intelligent online algorithms that can simplify the incoming trajectory data instantaneously.

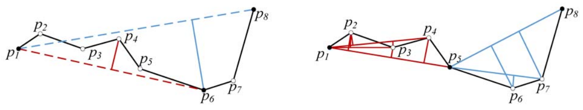
Most existing studies on trajectory simplification such as [2], [4], [11], [13], [15] aim at preserving the *position* information of trajectories. There is another method called delta compression [14] achieves good performance on trajectory simplification, whose main idea is to store the initial data point and then store the remaining data points as a sequence of successive deltas. Although [14] achieves lossless compression and high compression rate, it just works offline. Existing online simplification algorithms [7], [9], [12] also make the assumption that the goal should be to compress trajectories such that the position information captured in the simplified trajectories is "similar" to the position information captured in the raw trajectories. Recently, an alternative type of direction-preserving trajectory simplification [10] has been proposed with an attracting property. Long et al. showed that direction-preserving trajectory simplification can retain both direction and position information. In other words, if a simplification method is direction-preserving, it is also position-preserving.

Unfortunately, the algorithms in [10] were designed for the offline scenario when all the raw trajectory data are available. Recently, an online approach called *Angular-Deviation* [5] was proposed to address the direction-based trajectory simplification problem. It achieves extremely low running time, but its compression rate is not satisfactory. In this paper, we study an efficient method for solving the problem of direction-preserving trajectory simplification in an online fashion. We first introduce three baseline solutions, *Direction-Preserving Buffered Douglas-Peucker* (DPBDP), *Direction-Preserving Buffered Greedy Deviation* (DPBGD) and *Angular-Deviation*. DPBDP and DPBGD are extended from Douglas-Peucker [4] and Greedy Deviation (a variation of the generic sliding window algorithm [6]) respectively. The three baseline solutions cannot achieve good performance on both running time and compression rate, which are the two most important evaluation criteria of a trajectory simplification algorithm. To achieve a better tradeoff between high compression rate and low running time, we propose an *Interval* algorithm by using interval bounds, which can quickly determine the reservation of a new point without comparison with the historical points.

Our contributions can be summarized as follows:

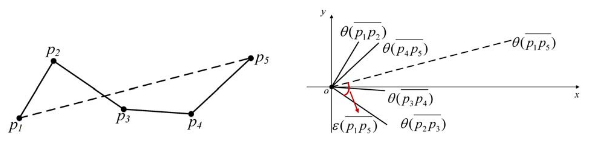
- We study a new problem of trajectory simplification which is direction-preserving and processed in an online fashion.
- Based on baseline solutions, we propose an improved *Interval* algorithm with error bound. *Interval* achieves $\mathcal{O}(n)$ time complexity and $\mathcal{O}(1)$ space complexity for the whole data stream.
- We conduct extensive experiments on real trajectory datasets. The results demonstrate superior performance of our approach on both running time and compression rate.

2375-0324/17 $31.00 © 2017 IEEE
DOI 10.1109/MDM.2017.17



(a) Douglas-Peucker. (b) Sliding Window.

Fig. 1. Existing Trajectory Simplification Methods.


p_1
p_2
p_3
p_4
p_5
y
x
o
$\theta(\overline{p_1p_2})$
$\theta(\overline{p_4p_5})$
$\theta(\overline{p_1p_5})$
$\theta(\overline{p_3p_4})$
$\theta(\overline{p_2p_3})$
$\varepsilon(\overline{p_1p_5})$

(a) Trajectory. (b) Error of Simplified Segment.

Fig. 2. Direction-Preserving Error Measurement.

The remainder of this paper is organized as follows. We review the related work in Section II. Three baseline solutions are introduced in Section III, and subsequently we present the proposed *Interval* algorithm in Section IV. We compare the performance of different algorithms in Section V and finally conclude in Section VI.

II. Related Work

In this section, we first introduce two representative trajectory simplification methods that can be applied on a small buffer of data. Then, we elaborate the direction-preserving error measurement.

A. Trajectory Simplification Techniques

Douglas-Peucker. As shown in Figure 1(a), Douglas-Peucker algorithm [2] replaces a raw trajectory by an approximate line segment ($\overline{p_1p_8}$). If the replacement does not meet the specified error requirement, it recursively partitions the raw trajectory into two sub-trajectories ($\{p_1, p_2, p_3, p_4, p_5, p_6\}$ and $\{p_6, p_7, p_8\}$) by selecting the point which contributes the largest error as the splitting point (p_6).

Sliding Window. The main idea of a generic sliding window algorithm [6] is to append the points to a growing sliding window and continue to grow the sliding window until the approximation error exceeds an predefined error tolerance. As illustrated in Figure 1(b), p_5 will be first reserved as the error for p_4 exceeds the error tolerance. Then the algorithm starts from p_5 and reserves p_8. Consequently, the final simplification results are $\{p_1, p_5, p_8\}$.

Both of the two methods aim at preserving position information, and Douglas-Peucker algorithm can only run offline. They cannot be directly used for direction-preserving trajectory simplification. In Section III, we will develop two baseline solutions extended from these two methods to support the simplification in the new scenario.

B. Direction-Preserving Error Measurement

Let $T = \{p_1, p_2, ..., p_n\}$ be a raw trajectory, we say that trajectory T' is a simplification of T if $T' = \{p_{s_1}, p_{s_2}, ..., p_{s_m}\}$ where $m \leq n$ and $1 = s_1 < s_2 < ... < s_m = n$. The direction-preserving error of simplification is the maximum value of angular difference between segment $\overline{p_ip_{i+1}}$ and corresponding simplified segment $\overline{p_{s_k}p_{s_{k+1}}}$. We use $\varepsilon(\overline{p_{s_k}p_{s_{k+1}}})$ to denote the error of a simplified segment $\overline{p_{s_k}p_{s_{k+1}}}$ as:

$$\varepsilon(\overline{p_{s_k}p_{s_{k+1}}}) = \max_{s_k \leq i < s_{k+1}} \Delta(\theta(\overline{p_ip_{i+1}}), \theta(\overline{p_{s_k}p_{s_{k+1}}})) \quad (1)$$

where$\Delta(\theta_1, \theta_2) = min\{|\theta_1 - \theta_2|, 2\pi - |\theta_1 - \theta_2|\}$. As shown in Figure 2, $\overline{p_1p_5}$ is a simplified segment, $\varepsilon(\overline{p_1p_5}) = \max_{1 \leq i < 5} \Delta(\theta(\overline{p_ip_{i+1}}), \theta(\overline{p_1p_5})) = \Delta(\theta(\overline{p_2p_3}), \theta(\overline{p_1p_5}))$.

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

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

In general, direction-preserving trajectory simplification is superior to position-based trajectory simplification [10]. Thus, our work focuses on addressing direction-preserving trajectory simplification in an online fashion.

III. Baseline Solutions

In this section, we present three baseline algorithms which are very natural solutions to the problem of online trajectory simplification, and then analyze the time complexity and space complexity of the three algorithms.

A. DPBDP Algorithm

We first introduce Direction-Preserving Douglas-Peucker (DDP) algorithm for offline direction-preserving trajectory simplification. The main idea of DDP is similar to the conventional DP algorithm [4], by replacing positional distance to angular difference. It recursively splits a trajectory into sub-trajectories by finding the points with the largest angular difference. Eventually, we can get $m-1$ sub-trajectories where m is the number of points in the simplified trajectory.

To apply the offline algorithm in the online scenario, a buffer is often needed [3], [8]. Based on it, we propose Direction-Preserving Buffered Douglas-Peucker (DPBDP) algorithm which is an online version of DDP. In DPBDP, the incoming points are first accumulated in a buffer, then the points are processed by applying DDP algorithm when the buffer is full. DPBDP has an inferior compression rate by the reason that at least two points need to be reserved whenever the buffer is full.

Complexity Analysis. There are $\mathcal{O}(n/B)$ times of calling DDP algorithm which causes $\mathcal{O}(B^2)$ time complexity in the worst case where n is the size of a trajectory and B denotes the buffer size. Therefore, the time complexity of DPBDP is $\mathcal{O}(B^2 \cdot (n/B))$=$\mathcal{O}(Bn)$. Obviously, the space complexity of DPBDP is $\mathcal{O}(B)$.

B. DPBGD Algorithm

Direction-Preserving Buffered Greedy Deviation (DPBGD) is a variation of the generic sliding window algorithm [6]. In DPBGD, we append the point to the end of the buffer whenever the point arrives if the buffer is not full, and calculate

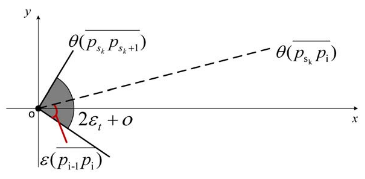
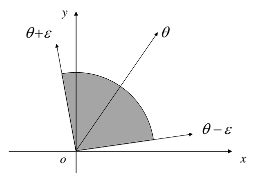
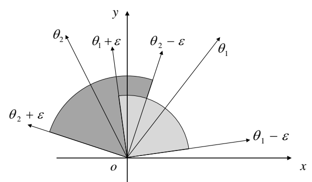

y
$\theta(\overline{p_{s_k}p_{s_{k+1}}})$
$\theta(\overline{p_{s_k}p_i})$
o
$2\varepsilon_t + o$
x
$\varepsilon(\overline{p_{i-1}p_i})$

Fig. 3. Illustration of Poor Compression Rate of Angular-Deviation.



(a) $Range(\theta, \varepsilon)$.



(b) $ComSub(\{\theta_1, \theta_2\}, \varepsilon)$.

Fig. 4. Illustrations of Definitions.

the error of the simplified segment defined by the start point in the buffer and the end point just arrived. If the error exceeds the tolerance, we reserve the last point, clear the buffer and start a new simplification at the last point. Otherwise, the process waits for the next incoming point. If the buffer is full, we reserve the end point of the buffer regardless of whether it leads to an exceeded error. DPBGD algorithm has a relatively high time complexity, since a complete calculation of the error is needed as long as a new point arrives.

Complexity Analysis. Consider the i^{th} incoming point in the buffer, we should calculate errors for $\mathcal{O}(i)$ times. In the worst case, calculations need to be executed for $\mathcal{O}(1+2+3+...+B)$=$\mathcal{O}(B^2)$ times. Therefore, the time complexity of DPBGD is $\mathcal{O}(B^2 \cdot (n/B))$=$\mathcal{O}(Bn)$. The space complexity of DPBGD is $\mathcal{O}(B)$ which is the same as DPBDP.

C. Angular-Deviation Algorithm

Angular-Deviation algorithm [5] is proposed recently for solving the problem of direction-preserving trajectory simplification in an online fashion. It has been proved in [5] that the error of a simplified segment $\overline{p_{s_k}p_i}$ (denoted by $\varepsilon(\overline{p_{s_k}p_i})$) must be smaller than the pre-defined error tolerance ε_t if the angular difference between $\overline{p_{s_k}p_{s_k+1}}$ and $\overline{p_{h-1}p_h}$ (denoted by $|\Delta(\overline{p_{s_k}p_{s_k+1}}, \overline{p_{h-1}p_h})|$) is less than $1/2\varepsilon_t$, where $p_{s_k+1} < h \leq i$.

To improve the efficiency of the computation, the Angular-Deviation algorithm makes the decision to reserve the point p_i if $|\Delta(\overline{p_{s_k}p_{s_k+1}}, \overline{p_{i-1}p_i})|$ is larger than $1/2\varepsilon_t$. The algorithm achieves low running time due to just several simple calculations are needed rather than an exact calculation of error as DPBGD does whenever a new point arrives.

However, the compression rate of Angular-Deviation is extremely low. As shown in Figure 3, in the same way as the lemma provided in [5], we can know that $\varepsilon(\overline{p_{s_k}p_i})$ must be larger than the pre-defined error tolerance ε_t if $|\Delta(\overline{p_{s_k}p_{s_k+1}}, \overline{p_{i-1}p_i})| > 2\varepsilon_t$. The point p_i which meets the Inequation 2 may not cause an unacceptable error, but the algorithm reserves the point p_i as long as $|\Delta(\overline{p_{s_k}p_{s_k+1}}, \overline{p_{i-1}p_i})| > 1/2\varepsilon_t$. More implementation details could be found in [5]. Note that Angular-Deviation algorithm does not require a buffer which is the same as the method we proposed in this paper.

$$1/2\varepsilon < |\Delta(\overline{p_{s_k}p_{s_k+1}}, \overline{p_{i-1}p_i})| \leq 2\varepsilon \qquad (2)$$

Complexity Analysis. By the reason of that only several simple addition operations are needed whenever a point arrives, the time complexity of Angular-Deviation is $\mathcal{O}(n)$. Due to the algorithm does not require a buffer, the space complexity is $\mathcal{O}(1)$.

IV. Approach Using Interval Bounds

In this section, we propose an algorithm which uses interval bounds to enhance the efficiency of trajectory simplification. Our key idea is to replace the exact calculation of error by a simple intersection operation on each point.

A. Definitions

Definition 1 (Direction Range). *Given a direction θ and an angle ε, the direction range denoted by $Range(\theta, \varepsilon)$ is:*

$$Range(\theta, \varepsilon) = [\theta - \varepsilon, \theta + \varepsilon],$$

where $[\theta_1, \theta_2]$ denotes the varying range of a vector originated from the origin when it is rotated anti-clockwise from θ_1 to θ_2.

As ε is normally smaller than $\pi/2$ in practice, an intersection between arbitrary $Range(\theta_i, \varepsilon)$ and $Range(\theta_j, \varepsilon)$ results in a single interval (in the circumstance that $\varepsilon > \pi/2$, an intersection between $Range(\theta_i, \varepsilon)$ and $Range(\theta_j, \varepsilon)$ may leads to two intervals).

Definition 2 (Common Subinterval). *Given a set of n directions $\{\theta_1, \theta_2,..., \theta_n\}$ and an angle ε, the common subinterval denoted by $ComSub(\{\theta_1, \theta_2, ..., \theta_n\}, \varepsilon)$ is defined as:*

$$ComSub(\{\theta_1, \theta_2, ..., \theta_n\}, \varepsilon) = \bigcap_{1 \leq i \leq n} Range(\theta_i, \varepsilon).$$

As shown in Figure 4(b), suppose $n = 2$, $ComSub(\{\theta_1, \theta_2\}, \varepsilon) = Range(\theta_1, \varepsilon) \cap Range(\theta_2, \varepsilon) = [\theta_2 - \varepsilon, \theta_1 + \varepsilon]$.

B. Lower and Upper Interval Bounds

Before we present our simplification algorithm, we first present three types of bound estimation that can facilitate the simplification process. Our online direction-preserving simplification algorithm relies on the following theorems.

Theorem 1 (Loose Upper Bound). *Given an angle ε and a set of n segments $\{seg_1, seg_2, ..., seg_n\}$, let $\theta(seg_i)$ denote the direction of seg_i. In the case that $ComSub(\{\theta(seg_1), \theta(seg_2), ..., \theta(seg_n)\}, \varepsilon)$ is **empty**, the maximum angular difference between the vector sum of n segments and any seg_i is larger than ε.*

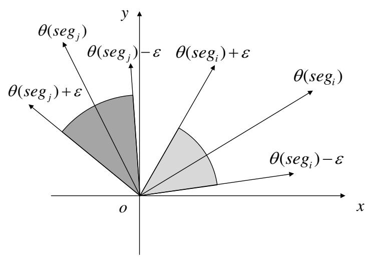

y
$\theta(seg_j)$
$\theta(seg_j)-\varepsilon$
$\theta(seg_i)+\varepsilon$
$\theta(seg_i)$
$\theta(seg_j)+\varepsilon$
$\theta(seg_i)-\varepsilon$
o
x

Fig. 5. Illustration of Theorem 1.

Proof: As shown in Figure 5, for two arbitrary directions $\theta(seg_i)$ and $\theta(seg_j)$, if $ComSub(\{\theta(seg_i), \theta(seg_j)\}, \varepsilon) = \emptyset$, there exists no direction θ that satisfies $\Delta(\theta(seg_i), \theta) \le \varepsilon$ and $\Delta(\theta(seg_j), \theta) \le \varepsilon$, where $\Delta(\theta_1, \theta_2)$ denotes the angular difference between θ_1 and θ_2. Suppose there exists such a direction θ, then θ must fall within $Range(\theta(seg.i), \varepsilon)$ and $Range(\theta(seg.j), \varepsilon)$ according to Definition 1, which implies $Range(\theta(seg.i)) \bigcap Range(\theta(seg.j))$ is non-empty. This leads to a contradiction. ■

Theorem 2 (Tight Lower Bound)**.** *Given an angle ε and a set of n segments $\{seg_1, seg_2, ..., seg_n\}$, if $ComSub(\{\theta(seg_1), \theta(seg_2), ..., \theta(seg_n)\}, \varepsilon)$ is* ***non-empty*** *and the vector sum of n segments falls in $ComSub$, the maximum angular difference between the vector sum of n segments and any seg_i is not larger than ε.*

Proof: Suppose $ComSub(\{\theta(seg_1), \theta(seg_2), ..., \theta(seg_n)\}, \varepsilon)$ equals to $[\theta_1, \theta_2]$, the arbitrary direction θ which falls within the range $[\theta_1, \theta_2]$ is limited in the interval $Range(\theta(seg_i), \varepsilon)$ for $1 \le i \le n$. Thus, the angular difference between θ and $\theta(seg_i)$ is not larger than ε, and the theorem is proved. ■

Theorem 3 (Tight Upper Bound)**.** *Given an angle ε and a set of n segments $\{seg_1, seg_2, ..., seg_n\}$, if $ComSub(\{\theta(seg_1), \theta(seg_2), ..., \theta(seg_n)\}, \varepsilon)$ is* ***non-empty****, and the vector sum of n segments is out of $ComSub(\{\theta(seg_1), \theta(seg_2), ..., \theta(seg_n)\}, \varepsilon)$, the maximum angular difference between the vector sum of n segments and any seg_i is larger than ε.*

Proof: The vector sum of n segments denoted by θ which falls outside the interval $ComSub(\{\theta(seg_1), \theta(seg_2), ..., \theta(seg_n)\}, \varepsilon)$ implies that there exists at least one such segment seg_i, for which the angular difference between θ and $\theta(seg_i)$ is larger than ε. Therefore, the maximum angular difference between simplified segment (the vector sum of n segments) and any segment is larger than ε. ■

C. Our Proposed Algorithm

The *Interval* algorithm is described in Algorithm 1. It starts by checking if p_i is the end point of the trajectory. When p_i is an immediate point, first the intersection between $Range(\theta(\overline{p_i p_{i+1}}), \varepsilon_t)$ and $ComSub(\{\theta(p_s), ..., \theta(p_{i-1})\}, \varepsilon_t)$ is calculated (lines 2-3). Then, the algorithm decides whether to reserve the point or not according to the new $ComSub(\{\theta(p_s), ..., \theta(p_i)\}, \varepsilon_t)$. If $ComSub$ is empty, the maximum angular difference between $\overline{p_s p_{i+1}}$ and any segment is larger than ε (by Theorem 1), so in this case, p_i should be reserved and the algorithm starts a new simplification from p_i (lines 4-7). When $ComSub$ is non-empty (lines 9-13), the algorithm reserves p_i if $\theta(\overline{p_s p_{i+1}})$ is out of $ComSub$ which implies $\varepsilon(\overline{p_s p_{i+1}})$ is larger than the error tolerance (by Theorem 3). Otherwise, if $\theta(\overline{p_s p_{i+1}})$ falls within $ComSub$, p_i should be discarded (by Theorem 2). Algorithm 1 achieves $\mathcal{O}(n)$ time and $\mathcal{O}(1)$ space complexity.

Algorithm 1 *Interval* Algorithm

```
Input:
    an error tolerance $\varepsilon_t$, the start point $p_1$ of the trajectory, the first point
    $p_s$ of a new sub-trajectory to be simplified
Output:
    the resulted simplification $T'$
Initialization: $p_s \leftarrow p_1$, $p_i \leftarrow p_2$, $p_{i+1} \leftarrow p_3$, $T' \leftarrow \{p_1\}$, $ComSub \leftarrow$
    $Range(\theta(\overline{p_1 p_2}), \varepsilon_t)$
 1: while $p_{i+1} \neq null$ do
 2:   $p_i.interval \leftarrow Range(\theta(\overline{p_i p_{i+1}}), \varepsilon_t))$;
 3:   $ComSub \leftarrow ComSub \cap pi.interval$;
 4:   if $ComSub = \emptyset$ then
 5:     $T' \leftarrow T' \bigcup \{p_i\}$;
 6:     $ComSub \leftarrow pi.interval$;
 7:     $p_s \leftarrow p_i$;
 8:   else
 9:     if $\theta(\overline{p_s p_{i+1}})$ out of $ComSub$ then
10:       $T' \leftarrow T' \bigcup \{p_i\}$;
11:       $ComSub \leftarrow pi.intervval$;
12:       $p_s \leftarrow p_i$;
13:     end if
14:   end if
15:   $p_i \leftarrow p_{i+1}$;
16:   wait for new incoming point $p_{i+1}$;
17: end while
18: $T' \leftarrow T' \bigcup \{p_i\}$;
19: return $T'$
```

In the sense that maintaining an intersection of intervals and determining whether a particular point should be the initiator of a new simplification of a subsequence, the *Interval* algorithm is similar to the optimal algorithm for polyline simplification in [1]. However, for our work of direction-preserving trajectory simplification, error is measured by angle, whereas in [1] the non-emptiness of the intersections is used.

D. Comparison with Baseline Algorithms

Through analyzing three baseline algorithms in Section III, we know that DPBGD achieves high compression rate while Angular-Deviation performs best in terms of time complexity. Thus, we focus on comparing *Interval* with DPBGD and Angular-Deviation.

1) Interval v.s. DPBGD: Both *Interval* and DPBGD belong to greedy algorithm. For the i^{th} incoming point p_i, *Interval* and DPBGD accurately judge that whether p_i leads to an unacceptable error, and thus the two algorithms achieve similar compression rate. The only difference is that, as Section III-B shows, to compute the error of any segment $\overline{p_{s_k} p_{s_{k+1}}}$, DPBGD uses Equation 1 which leads to $s_{k+1} - s_k$ calculations, and therefore a buffer is needed to avoid too many calculations. However, according to Algorithm 1, *Interval* performs only $\mathcal{O}(1)$ intersection operations instead of computing accurate error to make decision for p_i. Thus, *Interval* runs much faster than DPBGD, and it achieves slightly higher compression rate since it does not need a buffer.

2) Interval v.s. Angular-Deviation: For an incoming point p_i, both *Interval* and Angular-Deviation perform a simple operation to decide whether p_i should be discarded. Thus, the time complexities of them are low. We assume the fist point to be simplified is p_s. As Section III-C shows, Angular-Deviation reserves points which may not lead to an unacceptable error. However, our proposed *Interval* algorithm reserves p_i when $\varepsilon(\overline{p_s p_i})$ is exactly larger than the predefined error tolerance. Hence, *Interval* achieves higher compression rate than Angular-Deviation.

V. Experiments

We conducted experiments to evaluate the performance of our proposed *Interval* algorithm with the three baseline methods DPBDP, DPBGD and Angular-Deviation. All algorithms were implemented in Java and run on a Windows platform. Two real datasets are used: (1) Chengdu taxi dataset, sampling 14000 taxies in Chengdu for one month. Each trajectory contains 3038 points on average; (2) Geolife dataset [16], which records the outdoor movements of 182 users in a period of 5 years and contains 17621 trajectories. Overall, the trajectories of Chengdu taxi dataset have much larger variance in position and moving speed than the Geolife dataset.

A. Effect of Buffer Size on Running Time

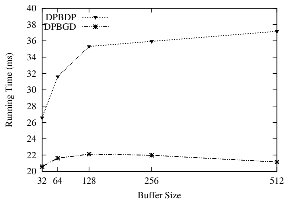
Since both of Angular-Deviation and *Interval* do not need buffers, in this experiment we study the effect of buffer size (i.e., B) on DPBDP and DPBGD. The values used for buffer size B are 32, 64, 128, 256, 512 data points (the error tolerance ε_t is fixed to be 1 and data size $|T|$ is fixed to be 5k). The running time results are shown in Figure 6.

For DPBDP, the running time increases with the buffer size B. Note that, when B is equal to the data size $|T|$, DPBDP can be reduced to the DDP algorithm introduced in Section III-A.

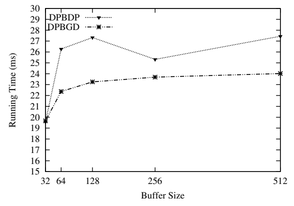
For DPBGD, when the buffer size is relatively small, the running time of DPBGD increases with B, but when B is larger than some value, the running time of DPBGD is stable. Consider that we give each point a number in ascending order when it is added in the buffer. For the i^{th} point in the buffer, it leads to $i+1$ times of calculations (2 times for direction and $i-1$ times for angular difference). That is, the larger number the point has, the more calculations are required. Consequently, the running time becomes longer. However, when buffer is large enough, the error of the simplified segment will always exceed ε_t before the buffer is full. In other words, the buffer has no effect on simplification, and thus the running time is not sensitive to buffer size B.

B. Effect of Data Size on Running Time

In this experiment, we study the effect of data size (i.e., $|T|$) on the performance of four algorithms (DPBDP, DPBGD,

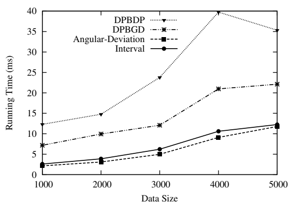


(a) Chengdu Taxi Dataset.

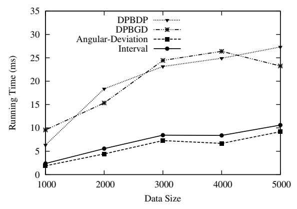


(b) Geolife Dataset.

Fig. 6. Effect of Buffer Size B.



(a) Chengdu Taxi Dataset.



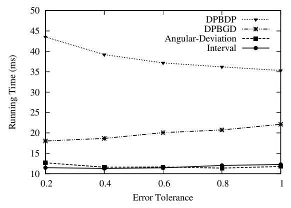
(b) Geolife Dataset.

Fig. 7. Effect of Data Size $|T|$.

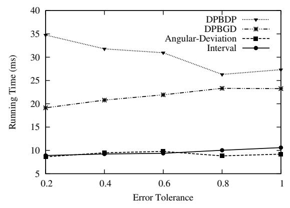
Angular-Deviation and *Interval*). The values used for data size $|T|$ are around 1k, 2k, 3k, 4k and 5k (the error tolerance ε_t is fixed to be 1 and the buffer size B is fixed to be 128). For each data size, we select a set of 10 trajectories, each of which has its size near to this value. Then, the average experimental results on these trajectories are shown in Figure 7. We observe that the running time of each algorithm increases almost linearly with data size. This is because the time complexities of DPBDP and DPBGD are proportional to data size ($\mathcal{O}(Bn)$), and both of Anguar-Deviation and *Interval* achieve $\mathcal{O}(n)$ time complexities. Besides, Angular-Deviation is the fastest among the four algorithms, and the performance of *Interval* is equally ideal by virtue of interval bounds to avoid all exact calculations of error. The performance of DPBDP and DPBGD are unsatisfactory.

C. Effect of Error Tolerance on Running Time

In this experiment, we study the effect of error tolerance ε_t on the performance of four algorithms. The values used for ε_t are 0.2, 0.4, 0.6, 0.8 and 1 in radians and we report the results in Figure 8 (buffer size B is set to be 128 and data size $|T|$ is fixed to be 5k). As shown in Figure 8, both of Angular-Deviation and *Interval* achieve the best performance. For *Interval*, the algorithm replaces the exact calculation of error by a simple intersection operation on each point. For Angular-Deviation, whatever ε_t is, the algorithm just needs to calculate the direction of each segment and the angular difference, so the running time of Angular-Deviation is low. It is worth noting that the impacts of ε_t on DPBDP and DPBGD are quite different. When ε_t becomes larger, the running time of DPBDP decreases, but when ε_t increases, DPBGD needs more time to simplify which is contrast to DPBDP. The reason is that, for DPBDP the number of segments leading to a split is reduced, but for DPBGD when ε_t becomes larger, the buffer

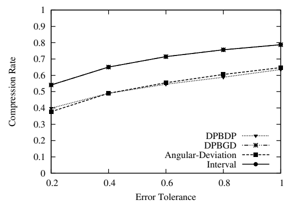


(a) Chengdu Taxi Dataset.

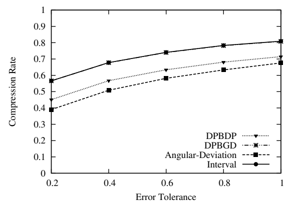


(b) Geolife Dataset.

Fig. 8. Effect of Error Tolerence ε_t.



(a) Chengdu Taxi Dataset.



(b) Geolife Dataset.

Fig. 9. Compression Rate.

maintained by DPBGD is more likely to be full which causes more calculations.

D. *Effect of Error Tolerance on Compression Rate*

In this experiment, we compare the compression rate which is a key performance indicator in trajectory simplification of four solutions with increasing error tolerance ε_t, and the results are shown in Figure 9 (buffer size B is fixed to be 128 and data size $|T|$ is fixed to be 5k). The compression rate β is defined as $\beta = (|T| - |T'|)/|T|$, where T represents the raw trajectory and T' represents the result of simplification. For the value of β, higher is better. When the value of error tolerance ε_t becomes larger, compression rates of four algorithms all increase. That is because the number of segments which cause the error of simplification exceeds ε_t decreases.

The performance of Angular-Deviation is the worst. The reason is that, the algorithm does not calculate the exact error of a simplified segment, and it reserves the points with accumulated angular deviation larger than $1/2\varepsilon_t$, which just probably do not lead to an error that exceeds ε_t (Section III-C).

The performance of DPBDP is also unsatisfactory just like Angular-Deviation, since for every B points in the buffer, besides the points that cause unacceptable errors there are at least two points to be reserved.

The *Interval* algorithm achieves the best performance, and the compression rate of DPBGD is slightly worse than *Interval*. The reason is that, both *Interval* and DPBGD reserve a point p_i which leads to an unacceptable error whenever its arrives, but DPBGD needs to reserve the last point in the buffer and starts a new simplification whenever buffer is full and *Interval* does not maintain a buffer.

VI. Conclusion

This paper studies a novel problem of online direction-preserving trajectory simplification. We first introduce three baseline solutions named DPBDP, DPBGD and Angular-Deviation. DPBDP and DPBGD are extended from Douglas-Peucker and Greedy Deviation algorithms respectively, and both of them achieve high running time. Among the three baseline algorithms, DPBDP performs poor on both running time and compression rate. DPBGD achieves highest compression rate but runs extremely slowly. The running time of Angular-Deviation is lowest, but its performance on compression rate is unsatisfactory.

To achieve a better tradeoff between high compression rate and low running time, an *Interval* algorithm with error bound guarantee is proposed by using interval bounds. Extensive experiments show that the *Interval* algorithm achieves superior performance on both running time and compression rate.

Acknowledgments

This work is supported by the National Nature Science Foundation of China (grants No. 61672133, No. 61602087 and No. 61632007), and the Fundamental Research Funds for the Central Universities (grants No. ZYGX2015J058 and No. ZYGX2014Z007).

References

[1] W. S. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimum error. *Int. J. Comput. Geometry Appl.*, 6(1):59–77, 1996.
[2] 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. *The Canadian Cartographer*, 10(2):112–122, 1973.
[3] O. Ghica, G. Trajcevski, O. Wolfson, U. Buy, P. Scheuermann, F. Zhou, and D. Vaccaro. Trajectory data reduction in wireless sensor networks. *IJNGC*, 1(1), 2010.
[4] P. S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. Technical report, Carnegie Mellon University, 1997.
[5] B. Ke, J. Shao, Y. Zhang, D. Zhang, and Y. Yang. An online approach for direction-based trajectory compression with error bound guarantee. In *Web Technologies and Applications - 18th Asia-Pacific Web Conference, APWeb 2016, Suzhou, China, September 23-25, 2016. Proceedings, Part I*, pages 79–91, 2016.
[6] E. J. Keogh, S. Chu, D. M. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. In *ICDM*, pages 289–296, 2001.
[7] A. Kolesnikov. Efficient online algorithms for the polygonal approximation of trajectory data. In *MDM*, pages 49–57, 2011.
[8] R. Lange, F. Dürr, and K. Rothermel. Efficient real-time trajectory tracking. *VLDB J.*, 20(5):671–694, 2011.
[9] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, and R. Jurdak. Bounded quadrant system: Error-bounded trajectory compression on the go. In *ICDE*, pages 987–998, 2015.
[10] C. Long, R. C. Wong, and H. V. Jagadish. Direction-preserving trajectory simplification. *PVLDB*, 6(10):949–960, 2013.
[11] N. Meratnia and R. A. de By. Spatiotemporal compression techniques for moving point objects. In *EDBT*, pages 765–782, 2004.
[12] J. Muckell, J. Hwang, V. Patil, C. T. Lawson, F. Ping, and S. S. Ravi. SQUISH: an online approach for GPS trajectory compression. In *COM.Geo*, pages 13:1–13:8, 2011.
[13] J. Muckell, P. W. Olsen, J. Hwang, C. T. Lawson, and S. S. Ravi. Compression of trajectory data: a comprehensive evaluation and new approach. *GeoInformatica*, 18(3):435–460, 2014.
[14] A. Nibali and Z. He. Trajic: An effective compression system for trajectory data. *IEEE Trans. Knowl. Data Eng.*, 27(11):3138–3151, 2015.
[15] M. Potamias, K. Patroumpas, and T. K. Sellis. Sampling trajectory streams with spatiotemporal criteria. In *SSDBM*, pages 275–284, 2006.
[16] Y. Zheng, L. Zhang, X. Xie, and W. Ma. Mining interesting locations and travel sequences from GPS trajectories. In *Proceedings of the 18th International Conference on World Wide Web, WWW 2009, Madrid, Spain, April 20-24, 2009*, pages 791–800, 2009.