

Error Bounded Line Simplification Algorithms for Trajectory Compression: An Experimental Evaluation

ANONYMOUS

Nowadays, various sensors are collecting, storing and transmitting tremendous trajectory data, and it is well-known that the storage, network bandwidth and computing resources could be heavily wasted if raw trajectory data is directly adopted. Line simplification algorithms are effective approaches to attacking this issue by compressing data points in a trajectory to a set of continuous line segments, and are commonly used in practice. In this article, we first classify the error bounded line simplification algorithms into different categories, and review each category of algorithms. We then study the data aging problem of line simplification algorithms and distance metrics from the views of aging friendliness and aging error, and provide a full picture of the data aging problem. Finally, we present a systematic experimental evaluation of representative error bounded line simplification algorithms, including both compression optimal and sub-optimal methods, in terms of commonly adopted perpendicular Euclidean, synchronous Euclidean and direction-aware distances. Using real-life trajectory datasets, we systematically evaluate and analyze the performance (compression ratio, average error, running time, aging friendliness and query friendliness) of error bounded line simplification algorithms with respect to distance metrics, trajectory sizes and error bounds. Our study reveals the characteristics of error bounded line simplification algorithms, which lead to guidelines for practitioners to choose appropriate algorithms and distance metrics for specific applications.

ACM Reference Format:

Anonymous. 2020. Error Bounded Line Simplification Algorithms for Trajectory Compression: An Experimental Evaluation. 1, 1 (April 2020), 35 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

With the increasing popularity of GPS sensors on various mobile devices, such as smart-phones, on-board diagnostics, personal navigation devices and wearable smart devices, trajectory data has been continuously growing. Further, sampling rates are also improved for acquiring more accurate position information, which leads to longer trajectories as well. Thus, transmitting and storing raw trajectory data consume a large amount of network, storage and computing resources [2, 5, 6, 12, 16, 17, 21, 25, 26, 28, 31, 32], and trajectory compression techniques [2, 5–7, 9, 12, 16, 17, 21, 25, 26, 31, 33, 35] have been developed to alleviate this situation.

Due to the limitations (poor compression ratio and data reconstruction overhead) of lossless compression, lossy compression techniques have become the mainstream of trajectory compression [15, 41]. Quite a few lossy trajectory compression techniques, most notably the piece-wise line simplification [2, 5–7, 9, 12, 16, 25, 33] solving the *min-#* problem [4, 10, 27], have been developed. The idea of piece-wise line simplification (LS) comes from computational geometry, whose target is to approximate a fine piece-wise linear curve with a coarse one (whose corresponding data points are a subset of the original one), such that the maximum distance of the former to the latter is bounded by a user specified threshold. Its wide usage is due to its beneficial features: (a) simple

Author's address: Anonymous.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

© 2020 Association for Computing Machinery.

XXXX-XXXX/2020/4-ART \$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

and easy to implement, (b) no need of extra knowledge and suitable for freely moving objects [28], and (c) bounded errors with good compression ratios.

Algorithm taxonomy. LS algorithms fall into two categories: *compression optimal* and *compression sub-optimal* algorithms. *Compression optimal* methods [4, 10] are to find the minimum number of points or segments to represent the original polygonal lines *w.r.t.* an error bound ϵ , by transforming the problem to search for the shortest path of a graph built from the original trajectory. The optimal LS algorithms have relatively high time/space complexities which make them impractical for large trajectory data. Hence, *compression sub-optimal* algorithms have been developed and/or introduced for trajectory compression, and they achieve better efficiency at an expense of outputting a little more data points. By the applied distance checking policies, compression sub-optimal algorithms can further be classified into batch, online and one-pass algorithms.

- (1) *Batch algorithms* are divided into top-down methods, *e.g.*, Douglas-Peucker (DP) [7, 21] and bottom-up methods, *e.g.*, Theo-Pavlidis (TP) [27], and apply global distance checking policies such that all trajectory points need to be loaded before starting compression such that a point may be checked multiple times to compute its distance to the corresponding line segments.
- (2) *Online algorithms*, such as OPW [21], SQUISH-E [24] and BQS [16], apply local distance checking policies, and need not have the entire trajectory ready before compressing. They restrict the checking within a window/buffer, but may still check a point multiple times during the process.
- (3) *One-pass algorithms*, such as OPERB [15], SIPED [8, 34, 39, 42], CISED [14], Intersect [17] and Interval [11], apply better local distance checking policies, which even do not need a window or buffer for the previously read points, and process each point in a trajectory once and only once.

Distance metrics. Trajectory simplification algorithms are closely coupled with distance metrics, and different techniques are typically needed for different distance metrics. We consider three widely adopted metrics: *perpendicular Euclidean distances* (PED), *synchronous Euclidean distances* (SED) and *direction-aware distances* (DAD).

Originally, LS algorithms adopt PED, the shortest Euclidean distance from a point to a line segment, as the distance metric. It brings good compression ratios [2, 6, 7, 9, 16, 25, 33] at a cost of losing temporal information of trajectories. (1) SED was then introduced to preserve the temporal information [2, 21]. The SED of a point to a line segment is the Euclidean distance between the point and its *synchronized point* *w.r.t.* the line segment, the expected position of the moving object on the line segment at the same time *with an assumption that the object moves straightly along the line segment at a uniform speed* [2]. It kindly supports applications such as spatio-temporal queries [14, 21], or in other words, it is *query friendly*, *i.e.*, a spatio-temporal query like *where_at* [2] on a simplified trajectory will return an expected point that has a distance less than the error bound used in the simplification algorithm. (2) DAD was introduced to preserve the direction information of moving objects [17, 41], and was initially called the *direction-based measurement* in [17]. It is important for applications such as trajectory clustering and direction-based query processing [17, 18]. Note that it is easy to know that given the same input, LS algorithms using SED or DAD may produce more line segments than using PED. However, the use of SED and DAD further preserves temporal and direction information, respectively.

Motivations. Empirical studies of trajectory compression algorithms have been conducted [23, 25, 38]. However, they only discuss a small number of algorithms. The very recent study [41] does evaluate a wide range of trajectory simplification algorithms. However, it provides an experimental study on compression errors and spatio-temporal query analyses only, and important aspects of trajectory simplification (compression ratios, running time and aging friendliness) are not systematically studied. Besides, the simplified trajectories are stored in data stores. As time goes

Table 1. Error bounded trajectory simplification algorithms

Category	Algorithms	PED	SED	DAD	Time	Space	Rep	Key Ideas
optimal	Optimal [10]	✓	✓	✓	$O(n^3)$	$O(n^2)$	✓	reachability graph
	OptPED [4]	✓	✗	✗	$O(n^2)$	$O(n^2)$		reachability graph and sector intersection
	SP [17]	✗	✗	✓	$O(n^2)$	$O(n^2)$		reachability graph and range intersection
sub-optimal	Ramer [29]	✓	✓	✓	$O(n^2)$	$O(n)$		top-down
	DP [7, 21]	✓	✓	✓	$O(n^2)$	$O(n)$	✓	top-down
	TP [27]	✓	✓	✓	$O(n^2/K)$	$O(n)$	✓	bottom-up
	OPW [21]	✓	✓	✓	$O(n^2)$	$O(n)$	✓	top-down in opening window
	BQS [16]	✓	✗	✗	$O(n^2)$	$O(n)$	✓	top-down, window and convex hull
	SWAB [12]	✓	✓	✓	$O(n * Q)$	$O(Q)$		bottom-up in sliding window
	SQUISH-E [25]	✗	✓	✗	$O(n \log Q)$	$O(Q)$	✓	bottom-up and priority queue
	RW [30]	✓	✗	✗	$O(n)$	$O(1)$		strip
	LDR [13, 37]	✗	✓	✗	$O(n)$	$O(1)$		linear dead reckoning
one-pass	OPERB [15]	✓	✗	✗	$O(n)$	$O(1)$	✓	fitting function
	SIPED [8, 42]	✓	✗	✗	$O(n)$	$O(1)$	✓	sector intersection
	CISED [14]	✗	✓	✗	$O(n)$	$O(1)$	✓	spatio-temporal cone intersection
	Intersect [17]	✗	✗	✓	$O(n)$	$O(1)$	✓	range intersection with $\frac{\epsilon}{2}$ -range
	Interval [11]	✗	✗	✓	$O(n)$	$O(1)$	✓	range intersection with ϵ -range

Here, (1) ‘Rep’ means “Representative”, K is the number of the final segments of a trajectory and $|Q|$ is the size of a buffer/window, and (2) batch and top-down algorithms DP [7, 21] and Ramer [29] are aging friendly, while other algorithms are not.

by, less precision may become acceptable for old trajectories [2], and they need to be further simplified to coarse trajectories to save storage. However, can these simplified trajectories be further compressed by LS algorithms to coarse trajectories while still having bounded errors w.r.t. the original trajectories? This is referred to as the *data aging* problem, which is initially introduced and partially discussed for algorithms Optimal and DP only in [2]. That is to say, a systematic experimental evaluation of line simplification algorithms for trajectory compression remains necessary for practitioners to decide the appropriate algorithms and distance metrics for specific applications.

Contributions. In this article, we conduct a systematic experimental evaluation and analyses of the mainstream error bounded trajectory simplification algorithms for large-scale trajectory data.

(1) We classify the error bounded LS algorithms into different categories, review each category of algorithms, and systematically evaluate the representative algorithms of each category. Table 1 summarizes the algorithms that consist of optimal and sub-optimal algorithms, and the latter are further classified into batch, online and one-pass algorithms. Note that online and one-pass algorithms are typically designed for a specific distance metric (PED, SED or DAD) only.

(2) We study the data aging problem of LS algorithms and distance metrics from the views of aging friendliness and aging error, and prove that only algorithms running in both batch and top-down manners and using PED and/or SED are aging friendly, having an error bound of $\max\{\epsilon_1, \epsilon_2\}$ w.r.t. the original trajectory, and other algorithms have an error bound of $\epsilon_1 + \epsilon_2$ for data aging. These together provide a full picture of the data aging problem.

(3) Using real-life trajectory datasets, we systematically evaluate and analyze the performance (compression ratio, average error, running time, aging friendliness and query friendliness) of error bounded line simplification algorithms with respect to distance metrics, trajectory sizes and error bounds. Our study reveals the characteristics of these algorithms, which lead to guidelines for practitioners to choose appropriate algorithms and distance metrics for specific applications.

Essentially, this study is a necessary complement to existing studies by providing a systematic evaluation and analyses of error bounded trajectory simplification algorithms. More specifically,

(1) for a fair running time analysis, all algorithms are (re)-implemented in Java, unlike [41] that reports running time of algorithms with different programming languages, (2) compression ratio analyses are not considered in [41], (3) variations of distance metrics are not studied in [41], (4) data aging of LS algorithms is not studied in [41], and (5) optimal algorithms using PED and SED and one-pass algorithms SIPED and CISED are not investigated in [41]. Indeed, algorithm SIPED is *completely overlooked* by existing trajectory compression studies as it is originally developed in fields of computational geometry and pattern recognition [8, 34, 39, 42], and we recently find that it can be easily adopted for trajectory compression with good performance.

Organization. Section 2 introduces basic concepts of trajectory simplification, Section 3 and Section 4 systematically review optimal and sub-optimal LS methods, respectively, Section 5 analyzes the data aging of LS algorithms and Section 6 reports and analyzes the experimental results, followed by conclusions in Section 7. Additional line simplification algorithms are discussed in the appendix.

2 PRELIMINARY

In this section, we introduce some basic concepts for trajectory simplification. For convenience, notations used are summarized in [Table 2](#).

Points. A data point is defined as a triple $P(x, y, t)$, which represents that a moving object is located at *longitude* x and *latitude* y at *time* t . Note that data points can be viewed as points in the x-y-t 3D Euclidean space.

Trajectory. A *trajectory* $\ddot{\mathcal{T}}[P_0, \dots, P_n]$ is a sequence of points in a monotonically increasing order of their associated time values (*i.e.*, $P_i.t < P_j.t$ for any $0 \leq i < j \leq n$). Intuitively, a trajectory is the path (or track) that a moving object follows through space as a function of time [22].

Directed line segment. A directed line segment (or line segment for simplicity) \mathcal{L} is defined as $\overrightarrow{P_s P_e}$, which represents the closed line segment that connects the start point P_s and the end point P_e . Note that here P_s and P_e are different points that may not in a trajectory $\ddot{\mathcal{T}}$.

For the projection of a directed line segment \mathcal{L} on an x-y 2D space, where x and y are the longitude and latitude, respectively, we also use $|\mathcal{L}|$ and $\mathcal{L}.\theta \in [0, 2\pi)$ to denote the length of \mathcal{L} in the x-y 2D space, and its angle with the x-axis of the coordinate system (x, y) . That is, the projection of a directed line segment $\mathcal{L} = \overrightarrow{P_s P_e}$ on an x-y 2D space is treated as a triple $(P_s, |\mathcal{L}|, \mathcal{L}.\theta)$.

Piece-wise line representation. A *piece-wise line representation* $\overline{\mathcal{T}}[\mathcal{L}_0, \dots, \mathcal{L}_m]$ ($0 < m \leq n$) of a trajectory $\ddot{\mathcal{T}}[P_0, \dots, P_n]$ is a sequence of continuous *directed line segments* $\mathcal{L}_i = \overrightarrow{P_{s_i} P_{e_i}}$ ($i \in [0, m]$) of $\ddot{\mathcal{T}}$ such that $\mathcal{L}_0.P_{s_0} = P_0$, $\mathcal{L}_m.P_{e_m} = P_n$ and $\mathcal{L}_i.P_{e_i} = \mathcal{L}_{i+1}.P_{s_{i+1}}$ for all $i \in [0, m - 1]$. Note that (1) each directed line segment in $\overline{\mathcal{T}}$ essentially represents a continuous sequence of data points in trajectory $\ddot{\mathcal{T}}$, (2) a piece-wise line representation is essentially a simplified trajectory, and (3) a point (x, y, t) on a spherical surface is projected on a 2D flat plane in the piece-wise line representation.

For trajectory simplification, three distance metrics are commonly used, namely, the *perpendicular Euclidean distance* (PED), the *synchronous Euclidean distance* [21] (SED) and the *direction-aware distance*[17, 41] (DAD). Consider a data point P and a directed line segment $\mathcal{L} = \overrightarrow{P_s P_e}$.

Perpendicular Euclidean distance. The perpendicular Euclidean distance $ped(P, \mathcal{L})$ of point P to line segment \mathcal{L} is $\min\{|PQ|\}$ for any point Q on $\overrightarrow{P_s P_e}$.

Synchronous Euclidean distance. The synchronous Euclidean distance $sed(P, \mathcal{L})$ of point P to line segment \mathcal{L} is $|\overrightarrow{PP'}|$ that is the Euclidean distance from P to its *synchronized data point* P' w.r.t. \mathcal{L} , where the synchronized data point P' w.r.t. \mathcal{L} is defined as follows: (a) $P'.x = P_s.x + c \cdot (P_e.x - P_s.x)$, (b) $P'.y = P_s.y + c \cdot (P_e.y - P_s.y)$ and (c) $P'.t = P.t$, where $c = \frac{P.t - P_s.t}{P_e.t - P_s.t}$.

Table 2. Summary of notations

Notations	Semantics
P	a data point
$\tilde{\mathcal{T}}$	a trajectory $\tilde{\mathcal{T}}$ is a sequence of data points
$\bar{\mathcal{T}}$	a piece-wise line representation of a trajectory $\tilde{\mathcal{T}}$
\mathcal{L}	a directed line segment
M	a distance metric of PED, SED or DAD
$ped(P, \mathcal{L})$	the perpendicular Euclidean distance of point P to line segment \mathcal{L}
$sed(P, \mathcal{L})$	the synchronous Euclidean distance of point P to line segment \mathcal{L}
$dad(\mathcal{L}_1, \mathcal{L}_2)$	the direction-aware distance of line segment \mathcal{L}_1 to line segment \mathcal{L}_2
ϵ	an error bound
\mathcal{A}	a line simplification algorithm

Synchronized points are essentially virtual points with the assumption that an object moves along a straight line segment from P_s to P_e with a uniform speed, i.e., the average speed $\frac{|\overrightarrow{P_s P_e}|}{P_e.t - P_s.t}$ between points P_s and P_e [2, 14]. Then the *synchronized point* P' of a point P w.r.t. the line segment $\overrightarrow{P_s P_e}$ is the expected position of the moving object on $\overrightarrow{P_s P_e}$ at time $P.t$, obtained by a linear interpolation [2]. More specifically, a synchronized point P'_i of P_i ($s \leq i < e$) w.r.t. the line segment $\overrightarrow{P_s P_e}$ is a point on $\overrightarrow{P_s P_e}$ satisfying $|\overrightarrow{P_s P'_i}| = \frac{P_i.t - P_s.t}{P_e.t - P_s.t} \cdot |\overrightarrow{P_s P_e}|$, which means that the object moves from P_s to P_e at an average speed $\frac{|\overrightarrow{P_s P_e}|}{P_e.t - P_s.t}$, and its position at time $P_i.t$ is the point P'_i on $\overrightarrow{P_s P_e}$ having a distance of $\frac{P_i.t - P_s.t}{P_e.t - P_s.t} \cdot |\overrightarrow{P_s P_e}|$ to P_s [2, 5, 14, 21, 41].

Direction-aware distance. The direction-aware distance $dad(\mathcal{L}_1, \mathcal{L}_2)$ is the direction deviation from \mathcal{L}_1 to \mathcal{L}_2 , i.e., $\Delta(\mathcal{L}_1.\theta, \mathcal{L}_2.\theta) = \min\{|\mathcal{L}_1.\theta - \mathcal{L}_2.\theta|, 2\pi - |\mathcal{L}_1.\theta - \mathcal{L}_2.\theta|\}$. Note DAD differs from PED and SED in that it is a measure of angle changes, rather than Euclidean distances, and the temporal information is also lost when using DAD.

We illustrate these notations with examples.

Example 1: Consider Figure 1, in which (1) $\tilde{\mathcal{T}}[P_0, \dots, P_{10}]$ is a trajectory having 11 data points, (2) the set of two continuous line segments $\{\overrightarrow{P_0 P_4}, \overrightarrow{P_4 P_{10}}\}$, the set of four continuous line segments $\{\overrightarrow{P_0 P_2}, \overrightarrow{P_2 P_4}, \overrightarrow{P_4 P_7}, \overrightarrow{P_7 P_{10}}\}$ and the set of three continuous line segments $\{\overrightarrow{P_0 P_4}, \overrightarrow{P_4 P_5}, \overrightarrow{P_5 P_{10}}\}$ are three piece-wise line representations of trajectory $\tilde{\mathcal{T}}$, (3) $ped(P_4, \overrightarrow{P_0 P_{10}}) = |\overrightarrow{P_4 P_4^*}|$, where P_4^* is the perpendicular point of P_4 w.r.t. line segment $\overrightarrow{P_0 P_{10}}$, (4) For P_4 , its synchronized point P'_4 w.r.t. $\overrightarrow{P_0 P_{10}}$ satisfies $\frac{|\overrightarrow{P_0 P'_4}|}{|\overrightarrow{P_0 P_{10}}|} = \frac{P_4.t - P_0.t}{P_{10}.t - P_0.t} = \frac{4-0}{10-0} = \frac{2}{5}$, (5) $sed(P_4, \overrightarrow{P_0 P_{10}}) = |\overrightarrow{P_4 P'_4}|$, $sed(P_2, \overrightarrow{P_0 P_4}) = |\overrightarrow{P_2 P'_2}|$ and $sed(P_7, \overrightarrow{P_4 P_{10}}) = |\overrightarrow{P_7 P'_7}|$, where points P'_4 , P'_2 and P'_7 are the synchronized points of P_4 , P_2 and P_7 w.r.t. line segments $\overrightarrow{P_0 P_{10}}$, $\overrightarrow{P_0 P_4}$ and $\overrightarrow{P_4 P_{10}}$, respectively. and (6) $dad(\overrightarrow{P_5 P_6}, \overrightarrow{P_0 P_{10}}) = \theta_{56}$ is the DAD of line segments $\overrightarrow{P_5 P_6}$ to $\overrightarrow{P_0 P_{10}}$. \square

Error bounded algorithms. Given a trajectory $\tilde{\mathcal{T}}[P_0, \dots, P_n]$ and a pre-specified bound ϵ , a trajectory simplification algorithm \mathcal{A} using PED (respectively, SED and DAD) produces a piece-wise line representation $\bar{\mathcal{T}}[\mathcal{L}_0, \dots, \mathcal{L}_m]$ ($0 < m \leq n$) such that for each $i \in [0, m]$, line segment $\mathcal{L}_i = \overrightarrow{P_{s_i} P_{e_i}}$ ($s_i < e_i$) approximately represents the sub-trajectory $\tilde{\mathcal{T}}'[P_{s_i}, \dots, P_{e_i}]$ of $\tilde{\mathcal{T}}$.

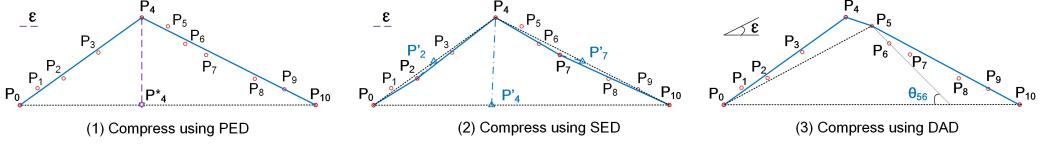


Fig. 1. A trajectory $\tilde{\mathcal{T}}[P_0, \dots, P_{10}]$ with 11 points is compressed by the Douglas–Peucker algorithm [7] using distance metrics PED, SED and DAD, respectively.

We say that algorithm \mathcal{A} using PED (respectively, SED and DAD) is error bounded by ϵ if for each point P_k ($k \in [0, n]$) in $\tilde{\mathcal{T}}$, there exists a line segment $\mathcal{L}_i = \overrightarrow{P_{s_i}P_{e_i}}$ in $\overline{\mathcal{T}}$ with $s_i \leq k \leq e_i$ ($0 \leq i \leq m$) such that the PED distance $ped(P_k, \mathcal{L}_i)$ (respectively the SED distance $sed(P_k, \mathcal{L}_i)$ and the DAD distance $dad(\overrightarrow{P_kP_{k+1}}, \mathcal{L}_i)$) is no more than ϵ .

Note that here we require that for all $i \in [0, m]$, points P_{s_i} and P_{e_i} of $\overline{\mathcal{T}}$ belong to the original trajectory $\tilde{\mathcal{T}}$. That is to say, all points in the simplified trajectory belong to the original trajectory.

3 COMPRESSION OPTIMAL ALGORITHMS

This section reviews the compression optimal LS algorithms that find the minimum number of points or segments to represent the original trajectory w.r.t. an error bound ϵ .

The naive optimal algorithm (Optimal) [10] first formulates it as a graph reachability problem, and solves the problem in $O(n^3)$ time, where n is the number of the original points of a trajectory. It is initially designed to support PED, but can easily be modified to support SED and DAD. By using *convex hull* [36] and *sector intersection* [20], faster optimal algorithms are proposed with an improved time complexity to $O(n^2 \log n)$. Further, [4] presents an optimal algorithm using PED achieves $O(n^2)$ time by using the *sector intersection* mechanism, and algorithm SP [17] is essentially an optimization of the original optimal algorithm using DAD that achieves $O(n^2)$ time. However, all the above optimization mechanisms do not support SED, and Optimal remains the best optimal solution for SED. As all the optimized algorithms have the same effectiveness using the same distance metric, and essentially work for small size trajectories only, we choose algorithm Optimal as the representative of optimal LS algorithms.

Given a trajectory $\tilde{\mathcal{T}}[P_0, \dots, P_n]$ and an error bound ϵ , algorithm Optimal [10] solves the optimal trajectory simplification problem in two steps: (1) it first constructs a reachability graph G of $\tilde{\mathcal{T}}$, and then (2) searches a shortest path from point P_0 to point P_n in graph G . The reachability graph of a trajectory $\tilde{\mathcal{T}}[P_0, \dots, P_n]$ w.r.t. an error bound ϵ is $G = (V, E)$, where (1) $V = \{P_0, \dots, P_n\}$, and (2) for any nodes P_s and $P_{s+k} \in V$ ($s \geq 0, k > 0, s + k \leq n$), edge $(P_s, P_{s+k}) \in E$ if and only if the distance of each point P_{s+i} ($0 < i < k$) to line segment $\overrightarrow{P_sP_{s+k}}$ is not greater than ϵ . Observe that in the graph G , (1) a path from nodes P_0 to P_n is a representation of trajectory $\tilde{\mathcal{T}}$. The path also reveals the subset of points of $\tilde{\mathcal{T}}$ used in the approximate trajectory, (2) the path length corresponds to the number of line segments in the approximate trajectory, and (3) a shortest path is an optimal representation of trajectory $\tilde{\mathcal{T}}$.

A straightforward way of constructing the reachability graph G needs to check for all pairs of points P_s and P_{s+k} whether the distances of all points $(P_{s+i}, 0 < i < k)$ to the line segment $\overrightarrow{P_sP_{s+k}}$ are less than ϵ . There are $O(n^2)$ pairs of points in the trajectory and checking the errors of all points P_{s+i} to a line segment $\overrightarrow{P_sP_{s+k}}$ takes $O(n)$ time. Thus, the construction step takes $O(n^3)$ time. Finding a shortest path takes no more than $O(n^2)$ time. Hence, the straightforward algorithm, i.e., Optimal, takes $O(n^3)$ time in total. For space complexity, it needs $O(n^2)$ space. Though the algorithm is initially developed using PED, it is easy to see that it also supports SED and DAD.

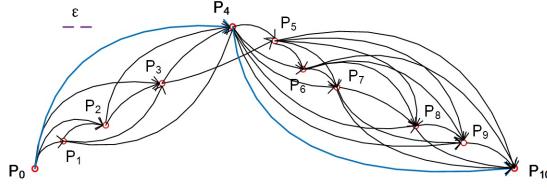


Fig. 2. Example of reachability graph of trajectory $\tilde{T}[P_0, \dots, P_n]$ whose shortest path is (P_0, P_4, P_{10}) .

Example 2: Figure 2 is an example of the Optimal algorithm using PED taking as input the trajectory \tilde{T} shown in Figure 1. The reachability graph of \tilde{T} is constructed and a shortest path with 2 edges is founded. At last, the algorithm outputs two line segments $\overrightarrow{P_0P_4}$ and $\overrightarrow{P_4P_{10}}$. \square

4 COMPRESSION SUB-OPTIMAL ALGORITHMS

This section reviews compression sub-optimal algorithms solving the *min-#* problem of trajectory simplification, as shown in the taxonomy of Section 1.

4.1 Batch Algorithms

Batch algorithms essentially apply global distance checking policies for trajectory simplification, and can be either top-down or bottom-up. Global checking policies enforce batch algorithms to have an entire trajectory first [21].

(1) Top-down algorithms recursively divide a trajectory into sub-trajectories until the stopping condition is met. Algorithms Ramer [29] and Douglas-Peucker (DP) [7] are similar, and support all the three distances PED, SED and DAD. An improved method of DP with a time complexity of $O(n \log n)$, based on *convex hulls*, is proposed in [9], which is the best DP based algorithm in terms of time complexities, and is designed for PED only, not for SED and DAD.

(2) Bottom-up algorithms are the natural complement of top-down ones, and they recursively merge adjacent sub-trajectories with the smallest distance, initially $n/2$ sub-trajectories for a trajectory with n points, until the stopping condition is met. Note that the distances of newly generated line segments are recalculated in each iteration. To our knowledge, Theo-Pavlidis (TP) [27] is the only bottom-up batch LS algorithm for trajectory simplification.

Note that, compared with top-down algorithms, bottom-up algorithms fit better for trajectories with lower sampling rates, as they typically need more rounds to merge smaller line segments into larger line segments. *Batch algorithms basically work for small and medium size trajectories, and we choose DP and TP that all support PED, SED and DAD as the representatives of batch LS algorithms.*

4.1.1 Algorithm Douglas-Peucker (DP) [7]. It is invented for reducing the number of points required to represent a digitized line or its caricature in the context of computer graphics and image processing. Originally, it uses PED, however, it can be easily extended to support SED and DAD.

Given a trajectory $\tilde{T}[P_0, \dots, P_n]$ and an error bound ϵ , algorithm DP uses the first point P_0 and the last point P_n of \tilde{T} as the start point P_s and the end point P_e of the first line segment $\mathcal{L}(P_0, P_n)$, then it calculates the distance $ped(P_i, \mathcal{L})$ for each point P_i ($i \in [0, n]$). If $ped(P_k, \mathcal{L}) = \max\{ped(P_0, \mathcal{L}), \dots, ped(P_n, \mathcal{L})\} \leq \epsilon$, then it returns $\{\mathcal{L}(P_0, P_n)\}$. Otherwise, it divides \tilde{T} into two sub-trajectories $\tilde{T}[P_0, \dots, P_k]$ and $\tilde{T}[P_k, \dots, P_n]$, and recursively compresses these sub-trajectories until the entire trajectory has been considered. The time complexity of DP is $\Omega(n)$ in the best case, but is $O(n^2)$ in the worst case.

Example 3: Consider the trajectory $\tilde{T}[P_0, \dots, P_{10}]$ shown in Figure 1. The DP Algorithm firstly creates $\overrightarrow{P_0P_{10}}$, then it calculates the distance of each point in $\{P_0, \dots, P_{10}\}$ to $\overrightarrow{P_0P_{10}}$. It finds that P_4 has

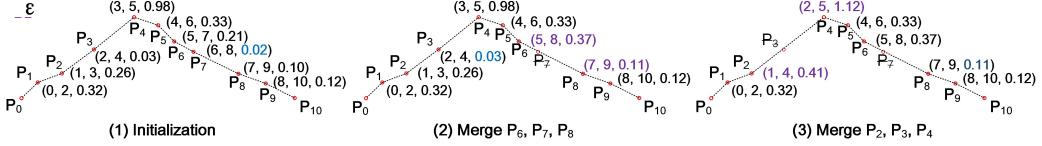


Fig. 3. The trajectory $\tilde{\mathcal{T}}[P_0, \dots, P_{10}]$ is compressed by the Theo-Pavlidis algorithm using PED to two line segments. The triple (i, j, cost) is the *cost* of merging the line segments $\overrightarrow{P_iP_j}$ and $\overrightarrow{P_jP_i}$.

the maximum distance to $\overrightarrow{P_0P_{10}}$, which is greater than ϵ . Then it goes to compress sub-trajectories $[P_0, \dots, P_4]$ and $[P_4, \dots, P_{10}]$, separately. When using SED (right), the sub-trajectory $[P_4, \dots, P_{10}]$ is further split to $[P_4, \dots, P_7]$ and $[P_7, P_{10}]$. Finally, the algorithm outputs two continuous directed line segments $\overrightarrow{P_0P_4}$ and $\overrightarrow{P_4P_{10}}$ when using PED, and three continuous directed line segments $\overrightarrow{P_0P_4}$, $\overrightarrow{P_4P_7}$ and $\overrightarrow{P_7P_{10}}$ when using SED. \square

4.1.2 Algorithm Theo-Pavlidis (TP) [27]. It initially employs the global checking policy to output disjoint line segments, and we slightly modify it to have continuous line segments.

Given a trajectory $\tilde{\mathcal{T}}[P_0, \dots, P_n]$ and an error bound ϵ , algorithm TP begins by creating the finest possible trajectory approximation: $[P_0, P_1], [P_1, P_2], \dots, [P_{n-1}, P_n]$, so that n segments are used to approximate the original trajectory. Next, for each pair of adjacent segments $[P_s, P_{s+j}]$ and $[P_{s+j}, P_{s+k}]$ ($0 \leq s < s+j < s+k \leq n$), the distance $\text{ped}(P_{s+i}, \overrightarrow{P_sP_{s+k}})$ of each point P_{s+i} ($0 < i < k$) to the line segment $\overrightarrow{P_sP_{s+k}}$, is calculated, and the max distance is saved and denoted as the *cost* of merging them. Then TP begins to iteratively merge the adjacent segment pair with the lowest cost until no cost is below ϵ . After the pair of adjacent segments $[P_s, P_{s+j}]$ and $[P_{s+j}, P_{s+k}]$ are merged to a new segment $[P_s, P_{s+k}]$, TP needs to recalculate the costs of the new segment with its preceding and successive segments, respectively. Algorithm TP runs in $O(n^2/K)$ time, where K is the number of the final segments. Similar to the DP algorithm, the TP algorithm originally supports PED, and it can be easily extended to support SED and DAD as well.

Example 4: Figure 3 is an example of the TP algorithm.

(1) Initially, 10 line segments are created, and for each pair of adjacent segments, the costs of merging them are calculated and saved. For example, the cost of merging $\overrightarrow{P_0P_1}$ and $\overrightarrow{P_1P_2}$ is $\text{ped}(P_1, \overrightarrow{P_0P_2}) = 0.32\epsilon$. (2) The cost of merging $\overrightarrow{P_6P_7}$ and $\overrightarrow{P_7P_8}$ is 0.02ϵ , which is the minimal value among all costs. Hence, $\overrightarrow{P_6P_7}$ and $\overrightarrow{P_7P_8}$ are merged to $\overrightarrow{P_6P_8}$. The cost of merging $\overrightarrow{P_5P_6}$ and $\overrightarrow{P_6P_8}$, and the cost of merging $\overrightarrow{P_6P_8}$ and $\overrightarrow{P_8P_9}$ are further updated to 0.37ϵ and 0.11ϵ , respectively. (3) $\overrightarrow{P_2P_3}$ and $\overrightarrow{P_3P_4}$ are merged to $\overrightarrow{P_2P_4}$. The cost merging $\overrightarrow{P_2P_4}$ and $\overrightarrow{P_4P_5}$, and the cost of merging $\overrightarrow{P_1P_2}$ and $\overrightarrow{P_2P_4}$ are also updated. (4) At last, the algorithm outputs two line segments $\overrightarrow{P_0P_4}$ and $\overrightarrow{P_4P_{10}}$. \square

4.2 Online Algorithms

Online LS algorithms adopt local checking policies by restricting the distance checking within a sliding or opening window such that there is no need to have the entire trajectory ready before compressing. That is, online algorithms essentially combine *batch algorithms* with *sliding or opening windows*, e.g., OPW [21] is a combination of top-down algorithm DP and opening windows while SWAB [12] is a combination of bottom-up algorithm TP and *sliding windows*. Though these algorithms support the three distance metrics PED, SED and DAD, they still have high time and/or space complexities [16]. To design more efficient online algorithms, techniques typically need to be designed closely coupled with distance metrics. Indeed, BQS [16] and SQUISH-E [25] propose to utilize convex hulls and priority queues, respectively, and they speed up trajectory simplification

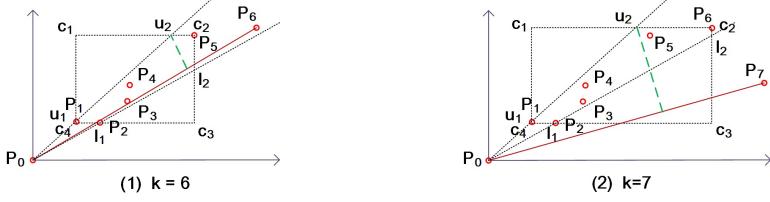


Fig. 4. Examples for algorithm BQS.

using PED and SED, respectively. To our knowledge, no specific techniques have been developed for DAD. Hence, we choose algorithms BQS, SQUISH-E and OPW as the representatives of online algorithms using PED, SED and DAD, respectively.

4.2.1 Algorithm OPW [21]. It combines the top-down and opening window strategies, and enforces the constrained global checking in the window. Like DP, it supports PED, SED and DAD.

Given a trajectory $\tilde{T}[P_0, \dots, P_n]$ and an error bound ϵ , algorithm OPW [21] maintains a window $W[P_s, \dots, P_k]$, where P_s and P_k are the start and end points, respectively. Initially, $P_s = P_0$ and $P_k = P_1$, and the window W is gradually expanded by adding new points one by one. OPW tries to compress all points in $W[P_s, \dots, P_k]$ to a single line segment $\mathcal{L}(P_s, P_k)$. If the distances $ped(P_i, \mathcal{L}) \leq \epsilon$ for all points P_i ($i \in [s, k]$), it simply expands W to $[P_s, \dots, P_k, P_{k+1}]$ ($k+1 \leq n$) by adding a new point P_{k+1} . Otherwise, it produces a new line segment $\mathcal{L}(P_s, P_{k-1})$, and replaces W with a new window $[P_{k-1}, \dots, P_{k+1}]$. The above process repeats until all points in \tilde{T} have been considered. The process is similar for SED and DAD. The time complexity of algorithm OPW remains in $O(n^2)$, the same as the DP algorithm.

4.2.2 Algorithm BQS Using PED [16]. It is essentially an efficiency optimized OPW algorithm [21], and reduces the running time by introducing convex hulls to pick out a certain number of points, which makes it specific for PED.

For a buffer W with sub-trajectory $[P_s, \dots, P_k]$, it splits the space into four quadrants. A buffer here is similar to a window in OPW [21]. For each quadrant, a rectangular bounding box is firstly created using the least and highest x and y values among points $\{P_s, \dots, P_k\}$, respectively. Then another two bounding lines connecting points P_s and P_h and points P_s and P_l are created such that lines $\overrightarrow{P_s P_h}$ and $\overrightarrow{P_s P_l}$ have the largest and smallest angles with the x -axis, respectively. Here $P_h, P_l \in \{P_s, \dots, P_k\}$. The bounding box and the two lines together form a convex hull. Each time a new point P_k is added to buffer W , BQS first picks out at most eight significant points from the convex hull in a quadrant. It calculates the distances of the significant points to line $\overrightarrow{P_s P_k}$, among which the largest distance d_u and the smallest distance d_l are an upper bound and a lower bound of the distances of all points in $[P_s, \dots, P_k]$ to line $\overrightarrow{P_s P_k}$. (1) If $d_l \geq \epsilon$, it produces a new line segment $\mathcal{L}(P_s, P_{k-1})$, and produces a new window $[P_{k-1}, \dots, P_k]$ to replace W . (2) If $d_u < \epsilon$, it simply expands buffer W to $[P_s, \dots, P_k, P_{k+1}]$ ($k+1 \leq n$) by adding a new point P_{k+1} . (3) Otherwise, it computes all distances $d(P_i, \mathcal{L}(P_s, P_k))$ ($i \in [s, k]$) as algorithm DP does. The time complexity of BQS remains $O(n^2)$. However, its simplified version FBQS has a linear time complexity by essentially avoiding case (3) to speed up the process.

Example 5: Figure 4 is an example of BQS. The bounding box $c_1c_2c_3c_4$ and the two lines $\overrightarrow{P_s P_h} = \overrightarrow{P_0 P_1}$ and $\overrightarrow{P_s P_l} = \overrightarrow{P_0 P_2}$ form a convex hull $u_1u_2c_2l_2l_1c_4$. BQS computes the distances of u_1, u_2, c_2, l_2, l_1 and c_4 to line $\overrightarrow{P_0 P_6}$ when $k = 6$ or to line $\overrightarrow{P_0 P_7}$ when $k = 7$. When $k = 6$, all these distances to $\overrightarrow{P_0 P_6}$ are less than ϵ , hence BQS goes on to the next point (case 2); When $k = 7$, the max and min distances to

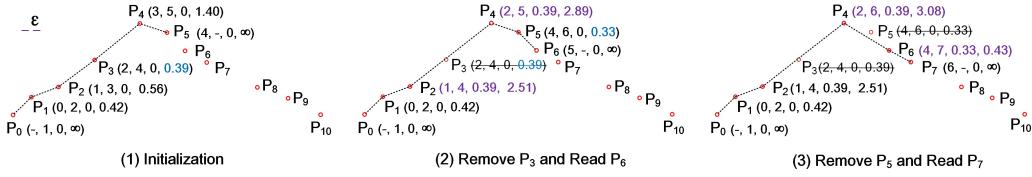


Fig. 5. The trajectory $\mathcal{T}[P_0, \dots, P_{10}]$ is compressed by the SQUISH-E algorithm using SED to five line segments. The size of Q is 6, and the data structure after point P is a tuple $(pre, suc, mmrrio, prio)$.

$\overrightarrow{P_0P_7}$ are larger and less than ϵ , respectively, and BQS needs to compress sub-trajectory $[P_0, \dots, P_7]$ along the same line as DP (case 3). \square

4.2.3 Algorithm SQUISH-E Using SED [25]. It is an online bottom-up algorithm that is specific for SED, and has two forms: SQUISH-E (λ) ensuring the compression ratio λ , and SQUISH-E (ϵ) ensuring the SED error bound ϵ . Here we adopt SQUISH-E (ϵ), as we focus on error bounded trajectory simplification.

Algorithm SQUISH-E optimizes algorithm TP with a doubly linked list Q . Each node in the list is a tuple $P(\text{pre}, \text{suc}, \text{prio}, \text{mnprio})$, where P is a trajectory data point, pre and suc are the predecessive and successive points of P , respectively, prio is the priority of P defined as an upper bound of the SED error that the removal of P introduces, and mnprio is the max priority of its predecessive and successive points removed from the list. Initially, trajectory points are loaded to Q one by one. At the same time, mnprio of each point is set to zero as no node has been removed from the list. Moreover, the priorities of points P_0 and $P_{|Q|-1}$ are set to ∞ , and the priority of point P_i ($0 < i < |Q| - 1$) is set to $\text{sed}(P_i, \text{pre}(P_i)\text{suc}(P_i))$. Then, SQUISH-E finds and removes a point P_j from Q that has the lowest priority $\text{prio}(P_j) < \epsilon$, and the properties mnprio of predecessor $\text{pre}(P_j)$ and successor $\text{suc}(P_j)$ are updated to $\max(\text{mnprio}(\text{pre}(P_j)), \text{prio}(P_j))$ and $\max(\text{mnprio}(\text{suc}(P_j)), \text{prio}(P_j))$, respectively. Next, the properties prio of $\text{pre}(P_j)$ and $\text{suc}(P_j)$ are further updated to $\text{mnprio}(\text{pre}(P_j)) + \text{sed}(\text{pre}(P_j), \text{pre}(\text{pre}(P_j))\text{suc}(P_j))$ and $\text{mnprio}(\text{suc}(P_j)) + \text{sed}(\text{suc}(P_j), \text{pre}(P_j)\text{suc}(\text{suc}(P_j)))$, respectively. After that, a new point is read to the list and the information of its predecessor in the list is updated. The above process repeated until that no points have a priority smaller than ϵ . SQUISH-E finds and removes a point from Q that has the lowest priority in $O(\log |Q|)$ time, where $|Q|$ denotes the number of points stored in Q . Thus, SQUISH-E runs in $O(n \log |Q|)$ time and $O(|Q|)$ space.

Example 6: Figure 5 is an example of SQUISH-E. (1) Initially, $|Q| = 6$ points are read to the list. The tuple $(pre, suc, mnprio, prio)$ for each point is initialized. For example, the tuple of P_1 is set to $(0, 2, 0, 0.42\epsilon)$, where 0.42ϵ is the SED from P_1 to $\overrightarrow{P_0P_2}$. (2) The priority of P_3 has the minimal value, thus, it is removed from the list. The $mnprio$ properties of P_2 and P_4 are updated to $\max\{mnprio(pre(P_3)), \text{prio}(P_3)\} = \max\{mnprio(P_2), \text{prio}(P_3)\} = \max\{0, 0.39\epsilon\} = 0.39\epsilon$, and $\max\{mnprio(P_4), \text{prio}(P_3)\} = 0.39\epsilon$, respectively. Furthermore, the $prio$ property of P_4 is updated to $mnprio(suc(P_j)) + sed(suc(P_j), pre(P_j)suc(suc(P_j))) = mnprio(P_4) + sed(P_4, \overrightarrow{P_2P_5}) = 0.39\epsilon + 2.50\epsilon = 2.89\epsilon$, and the $prio$ property of P_2 is updated to $mnprio(P_2) + sed(P_2, \overrightarrow{P_1P_4}) = 0.39\epsilon + 2.12\epsilon = 2.51\epsilon$. Then, P_6 is read, and the information of P_5 is updated. (3) P_5 is removed and P_7 is read to the list. (4) Finally, the algorithm outputs 5 line segments $\overrightarrow{P_0P_2}, \overrightarrow{P_2P_4}, \overrightarrow{P_4P_7}, \overrightarrow{P_7P_9}$, and $\overrightarrow{P_9P_{10}}$. \square

4.3 One-pass Algorithms

One-pass algorithms adopt local checking policies, and run in $O(n)$ time with an $O(1)$ space complexity. They are typically designed for specific distance metrics.

Reumann-Witkam (RW) [30] is a straightforward one-pass algorithm that builds a strip paralleling to the line connecting the first two points, then the points within this strip compose a section of the line. RW is fast, but has a poor compression ratio. Algorithm OPERB [15] recently improves RW by allowing dynamically adjustable strips, together with several detailed optimization techniques. There is also algorithm SIPED (*sector intersection*) that converts PED distance tolerances into angle tolerances to speed up the process, which is *completely overlooked* by existing trajectory compression studies, but can be easily adopted for trajectory compression, as it is originally developed in fields of computational geometry and pattern recognition [8, 34, 39, 42]. These algorithms are dedicatedly designed for PED. *Algorithms OPERB and SIPED have good compression ratios, and, hence, we choose them as the representatives of one-pass algorithms using PED.*

Algorithm Linear Dead Reckoning (LDR) for position tracking [13, 37] follows the similar routine as RW except that it uses SED and assumes a velocity \vec{v} for each section. It has poor compression ratios because both the value and the direction of velocity \vec{v} are pre-defined and fixed between two updates. Recently algorithm CISED [14] extends the *sector intersection* method SIPED from a 2D space to a spatio-temporal 3D space. These one-pass algorithms are dedicatedly designed for SED. *As algorithm CISED has a compression ratio close to algorithm DP using SED, we choose it as the representative of one-pass algorithms using SED.*

Direction range intersection approaches are similar to *sector intersection* methods except that they are designed for DAD, we choose Intersect [17] and Interval [11] as the representatives of one-pass algorithms using DAD.

4.3.1 Algorithm OPERB Using PED [15]. It designs a local distance checking method to dynamically adjust the direction of line segments to achieve an effective one-pass process.

Consider an error bound ϵ and a sub-trajectory $\tilde{\mathcal{T}}_s[P_s, \dots, P_{s+k}]$. OPERB dynamically maintains a directed line segment \mathcal{L}_i ($i \in [1, k]$), whose start point is fixed with P_s and its end point is identified (may not in $\{P_s, \dots, P_{s+i}\}$) to *fit* all the previously processed points $\{P_s, \dots, P_{s+i}\}$. The directed line segment \mathcal{L}_i is built by a function named *fitting function* \mathbb{F} , such that when a new point P_{s+i+1} is considered, only its distance to the directed line segment \mathcal{L}_i is checked, instead of checking the distances of all or a subset of data points of $\{P_s, \dots, P_{s+i}\}$ to $\mathcal{R}_{i+1} = \overrightarrow{P_s P_{s+i+1}}$ as the global distance checking does. During processing, if the distance of point P_{s+i} to the directed line segment \mathcal{L}_{i-1} is larger than the threshold, then a directed line segment, start from P_s , is generated and output; otherwise, the directed line segment \mathcal{L}_i is updated by the fitting function \mathbb{F} , as follows.

$$\begin{cases} [\mathcal{L}_i = \mathcal{L}_{i-1}] & \text{when } (|\mathcal{R}_i| - |\mathcal{L}_{i-1}|) \leq \frac{\epsilon}{4} \\ \left[\begin{array}{l} |\mathcal{L}_i| = j * \epsilon / 2 \\ \mathcal{L}_i.\theta = \mathcal{R}_i.\theta \end{array} \right] & \text{when } |\mathcal{R}_i| > \frac{\epsilon}{4} \text{ and } |\mathcal{L}_{i-1}| = 0 \\ \left[\begin{array}{l} |\mathcal{L}_i| = j * \epsilon / 2 \\ \mathcal{L}_i.\theta = \mathcal{L}_{i-1}.\theta + f(\mathcal{R}_i, \mathcal{L}_{i-1}) * \arcsin\left(\frac{\text{ped}(P_{s+i}, \mathcal{L}_{i-1})}{j * \epsilon / 2}\right) / j \end{array} \right] & \text{else} \end{cases}$$

where (a) $1 \leq i \leq k + 1$; (b) $\mathcal{R}_{i-1} = \overrightarrow{P_s P_{s+i-1}}$, is the directed line segment whose end point P_{s+i-1} is in $\tilde{\mathcal{T}}_s[P_s, \dots, P_{s+k}]$; (c) \mathcal{L}_i is the directed line segment built by fitting function \mathbb{F} to fit sub-trajectory $\tilde{\mathcal{T}}_s[P_s, \dots, P_{s+i}]$ and $\mathcal{L}_0 = \mathcal{R}_0$; (d) $j = \lceil (|\mathcal{R}_i| * 2/\epsilon - 0.5) \rceil$; (e) $f()$ is a sign function such that $f(\mathcal{R}_i, \mathcal{L}_{i-1}) = 1$ if the included angle $\angle(\mathcal{R}_{i-1}, \mathcal{R}_i) = (\mathcal{R}_i.\theta - \mathcal{L}_{i-1}.\theta)$ falls in the range of $(-2\pi, -\frac{3\pi}{2}]$,

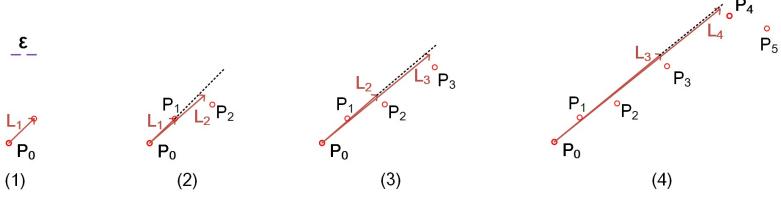


Fig. 6. The trajectory $\ddot{\mathcal{T}}[P_0, \dots, P_{10}]$ is compressed by the OPERB algorithm using PED to two line segments.

$[-\pi, -\frac{\pi}{2}], [0, \frac{\pi}{2}]$ and $[\pi, \frac{3\pi}{2}]$, and $f(\mathcal{R}_i, \mathcal{L}_{i-1}) = -1$, otherwise; (f) $\epsilon/2$ is a step length to control the increment of $|\mathcal{L}|$. Optimizations are developed to achieve better compression ratios.

Example 7: Figure 6 is a running example of the OPERB algorithm compressing the same trajectory $\ddot{\mathcal{T}}[P_0, \dots, P_{10}]$. (1) It takes P_0 as the start point, reads P_1 and sets $\mathcal{L}_1 = \overrightarrow{P_0P_1}$. (2) It reads P_2 . The distance from P_2 to \mathcal{L}_1 is less than the threshold, thus, it updates \mathcal{L}_1 to \mathcal{L}_2 by the fitting function \mathbb{F} . (3) It reads P_3 and P_4 , and updates \mathcal{L}_2 to \mathcal{L}_3 and \mathcal{L}_3 to \mathcal{L}_4 , respectively. (4) It reads P_5 . The distance from P_5 to \mathcal{L}_4 is larger than the threshold, thus, it outputs $\overrightarrow{P_0P_4}$ and starts the next section taking P_4 as the new start point. (5) The process continues until all points have been processed. At last, the algorithm outputs two continuous line segments $\overrightarrow{P_0P_4}$ and $\overrightarrow{P_4P_{10}}$. \square

4.3.2 Algorithm SIPED Using PED [8, 34, 39, 42]. It develops a concept of “Sector” [8, 34, 39, 42], which converts the distance tolerance into an angle change tolerance for checking points.

Given a sequence of points $[P_s, P_{s+1}, \dots, P_{s+k}]$ and an error bound ϵ , for the start data point P_s , any point P_{s+i} and $|\overrightarrow{P_sP_{s+i}}| > \epsilon$ ($i \in [1, k]$), there are two directed lines $\overrightarrow{P_sP_{s+i}^u}$ and $\overrightarrow{P_sP_{s+i}^l}$ such that $ped(P_{s+i}, \overrightarrow{P_sP_{s+i}^u}) = ped(P_{s+i}, \overrightarrow{P_sP_{s+i}^l}) = \epsilon$ and either $(\overrightarrow{P_sP_{s+i}^l} \cdot \theta < \overrightarrow{P_sP_{s+i}^u} \cdot \theta \text{ and } \overrightarrow{P_sP_{s+i}^u} \cdot \theta - \overrightarrow{P_sP_{s+i}^l} \cdot \theta < \pi)$ or $(\overrightarrow{P_sP_{s+i}^l} \cdot \theta > \overrightarrow{P_sP_{s+i}^u} \cdot \theta \text{ and } \overrightarrow{P_sP_{s+i}^u} \cdot \theta - \overrightarrow{P_sP_{s+i}^l} \cdot \theta < -\pi)$. Indeed, they form a sector $S(P_s, P_{s+i}, \epsilon)$ that takes P_s as the center point and $\overrightarrow{P_sP_{s+i}^u}$ and $\overrightarrow{P_sP_{s+i}^l}$ as the borderlines. There exists a data point Q such that for any data point P_{s+i} ($i \in [1, \dots, k]$), its perpendicular Euclidean distance to directed line $\overrightarrow{P_sQ}$ is not greater than the error bound ϵ if and only if the k sectors $S(P_s, P_{s+i}, \epsilon)$ ($i \in [1, k]$) share common data points other than P_s , i.e., $\bigcap_{i=1}^k S(P_s, P_{s+i}, \epsilon) \neq \{P_s\}$ [34, 39, 42]. Here, point Q may not belong to $\{P_s, P_{s+1}, \dots, P_{s+k}\}$. However, if Q must be a point selected from the original points, in other words, point P_{s+i} ($1 \leq i \leq k$) is chosen as Q , then for any point P_{s+j} ($j \in [1, \dots, i]$), its PED to line segment P_sP_{s+i} is not greater than the error bound ϵ if $\bigcap_{j=1}^i S(P_s, P_{s+j}, \epsilon/2) \neq \{P_s\}$, as pointed out in [42]. That is, *these sector intersection based algorithms can be easily adopted for trajectory compression*. In practice, it is a good choice to set the point Q as the point among all points in $[P_s, P_{s+1}, \dots, P_{s+k}]$ that has the longest distance to P_s .

The original SIPED uses a half sector, $\frac{\epsilon}{2}\mathcal{S}$, which may limit its compression performance. However, it can further be extended to a full $\epsilon\mathcal{S}$ by adding a constraint. That is, for any point P_{s+j} ($j \in [1, \dots, i]$), its PED to line segment $\overrightarrow{P_sP_{s+i}}$ is not greater than the error bound ϵ if $P_{s+i} \neq P_s$ and $P_{s+i} \in \bigcap_{j=1}^{i-1} S(P_s, P_{s+j}, \epsilon)$, i.e., P_{s+i} lives in the common intersection of preview full sectors.

Example 8: Figure 7 is a running example of algorithm SIPED ($\frac{\epsilon}{2}$) taking as input the same trajectory $\ddot{\mathcal{T}}[P_0, \dots, P_{10}]$. At the beginning, P_0 is the first start point, and points P_1, P_2, P_3 , etc., each has a narrow sector. For example, the narrow sector $S(P_0, P_3, \epsilon/2)$ takes P_0 as the center point and $\overrightarrow{P_0P_3^u}$ and $\overrightarrow{P_0P_3^l}$ as the borderlines. Because $\bigcap_{i=1}^4 S(P_0, P_{0+i}, \epsilon/2) \neq \{P_0\}$ and $\bigcap_{i=1}^5 S(P_0, P_{0+i}, \epsilon/2) = \{P_0\}$,

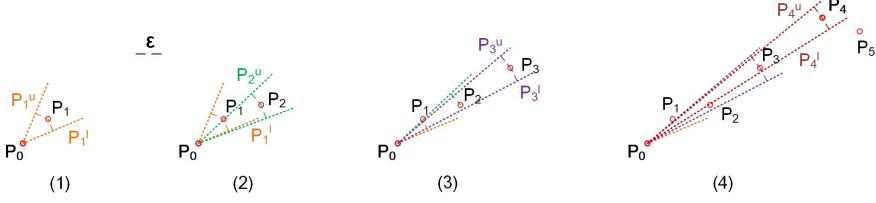
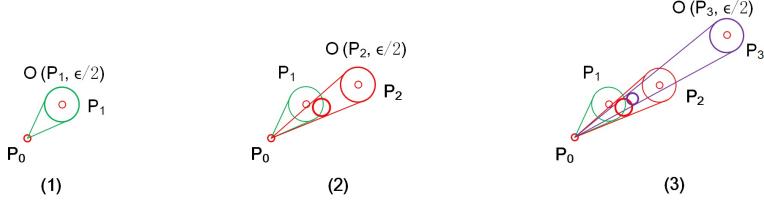
Fig. 7. The trajectory \tilde{T} is compressed by the sector intersection algorithm using PED to two line segments.

Fig. 8. A running example of the CISED algorithm. The points and the oblique circular cones are projected on an x-y space.

$\overrightarrow{P_0P_4}$ is output and P_4 becomes the start point of the next section. At last, the algorithm outputs two continuous line segments $\overrightarrow{P_0P_4}$ and $\overrightarrow{P_4P_{10}}$. \square

4.3.3 Algorithm CISED Using SED [14]. It develops an idea of *spatio-temporal cone* that extends the *sector intersection* method [8, 34, 39, 42] from a 2D space to a spatio-temporal 3D space.

Given a sub-trajectory $[P_s, \dots, P_{s+k}]$ and an error bound ϵ , any point P'_{s+i} ($0 < i \leq k$) on the plane $P.t - P_{s+i}.t = 0$ is a synchronized data point of P_{s+i} . For all P'_{s+i} in the plane satisfying $|P_{s+i}P'_{s+i}| \leq \epsilon$, they form a *synchronous circle* $O(P_{s+i}, \epsilon)$ of P_{s+i} with P_{s+i} as its center and ϵ as its radius. A spatio-temporal cone (or simply *cone*) of a data point P_{s+i} ($1 \leq i \leq k$) in \tilde{T}_s w.r.t. a point P_s and an error bound ϵ , denoted as $C(P_s, O(P_{s+i}, \epsilon))$, or C_{s+i} in short, is an oblique circular cone such that point P_s is its apex and the synchronous circle $O(P_{s+i}, \epsilon)$ is its base. Then, there exists a point Q such that $Q.t = P_{s+k}.t$ and $sed(P_{s+i}, \overrightarrow{P_sQ}) \leq \epsilon$ for each $i \in [1, k]$ if and only if $\bigcap_{i=1}^k C(P_s, O(P_{s+i}, \epsilon)) \neq \{P_s\}$. Like *sector intersection* methods, point Q may also not belong to $\{P_s, P_{s+1}, \dots, P_{s+k}\}$. If P_{s+i} ($1 \leq i \leq k$) is chosen as Q , then for any point P_{s+j} ($j \in [1, \dots, i]$), its SED to line segment $\overrightarrow{P_sP_{s+j}}$ is not greater than the error bound ϵ if $\bigcap_{j=1}^i C(P_s, O(P_{s+j}, \epsilon/2)) \neq \{P_s\}$ as pointed out in [14].

In addition, because these spatio-temporal cones have the same apex P_s , the checking of their intersection can be computed by a much simpler way, i.e., the checking of the intersection of cone projection circles on a plane, and a circle is further approximated with its m -edge inscribed regular polygon, whose intersection can be computed more efficiently.

The original CISED also uses a half cone, $\frac{\epsilon}{2}\text{-}C$. It can be extended to $\epsilon\text{-}C$ by adding such a constraint, i.e., P_{s+i} lives in the common intersection of preview full cones.

Example 9: Figure 8 shows a running example of algorithm CISED ($\frac{\epsilon}{2}$) for compressing the trajectory \tilde{T} in Figure 1. For convenience, we project the points and the oblique circular cones on an x-y space. (1) After initialization, the CISED algorithm reads point P_1 and builds a narrow oblique circular cone $C(P_0, O(P_1, \epsilon/2))$, taking P_0 as its apex and $O(P_1, \epsilon/2)$ as its base (green dash). The circular cone is projected on the plane $P.t - P_1.t = 0$, and the inscribe regular polygon \mathcal{R}_1 of the projection circle is returned. As \mathcal{R}^* is empty, \mathcal{R}^* is set to \mathcal{R}_1 . (2) The algorithm reads P_2 and builds $C(P_0, O(P_2, \epsilon/2))$ (red dash). The circular cone is also projected on the plane $P.t - P_1.t = 0$.

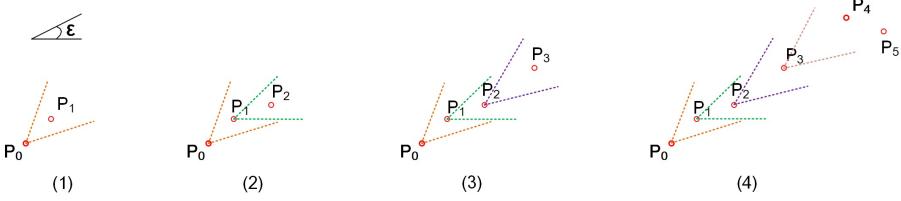


Fig. 9. The trajectory $\tilde{\mathcal{T}}$ is compressed by the interval algorithm using DAD to two line segments.

and the inscribe regular polygon \mathcal{R}_2 of the projection circle is returned. As $\mathcal{R}^* = \mathcal{R}_1$ is not empty, \mathcal{R}^* is set to the intersection of \mathcal{R}_2 and \mathcal{R}^* , which is $\mathcal{R}_1 \cap \mathcal{R}_2 \neq \emptyset$. (3) For point P_3 , the algorithm runs the same routine as P_2 until the intersection of \mathcal{R}_3 and \mathcal{R}^* is \emptyset . Thus, a line segment $\overrightarrow{P_0P_2}$ is generated, and the process of a new line segment is started, taking P_2 as the new start point and $P.t - P_3.t = 0$ as the new projection plane. (4) At last, the algorithm outputs four continuous line segments, i.e., $\{\overrightarrow{P_0P_2}, \overrightarrow{P_2P_4}, \overrightarrow{P_4P_7}, \overrightarrow{P_7P_{10}}\}$. \square

4.3.4 Algorithms Intersect [17] and Interval [11] Using DAD. It designs a *direction range* for each line segment connecting two neighboring points, then checks the common intersection of those *direction ranges* as a way similar to the *sector intersection* approach [8, 34, 39, 42].

Given a direction line segment \mathcal{L} and an angle ϵ , the *direction range* denoted by $range(\mathcal{L}, \theta, \epsilon)$ is $[\mathcal{L}, \theta - \epsilon, \mathcal{L}, \theta + \epsilon]$, which denotes the varying range of a directed line segment originated from the origin when it is rotated anti-clockwise from θ_1 to θ_2 [17]. The common intersection of *direction ranges* of directed line segments $\{\overrightarrow{P_sP_{s+1}}, \overrightarrow{P_{s+1}P_{s+2}}, \dots, \overrightarrow{P_{s+k-1}P_{s+k}}\}$ w.r.t. ϵ is $\bigcap_{i=1}^k Range(\overrightarrow{P_{s+i-1}P_{s+i}}, \theta, \epsilon)$.

Algorithm *Intersect* [17] uses a half range, and shows that if the common intersection $\bigcap_{i=1}^k range(\overrightarrow{P_{s+i-1}P_{s+i}}, \theta, \epsilon/2)$ is not empty, then the angular between $\overrightarrow{P_{s+i-1}P_{s+i}}$ and $\overrightarrow{P_sP_{s+k}}$ for all $i \in [1, k]$ is not larger than ϵ . Recently, algorithm *Interval* [11] extends *Intersect* from half to full ranges, by showing that if the common intersection $\bigcap_{i=1}^k range(\overrightarrow{P_{s+i-1}P_{s+i}}, \theta, \epsilon)$ is not empty and $P_s \overrightarrow{P_{s+k}} \cdot \theta$ falls in the common intersection, then the angular between $\overrightarrow{P_{s+i-1}P_{s+i}}$ and $\overrightarrow{P_sP_{s+k}}$ for all $i \in [1, k]$ is not larger than ϵ .

Example 10: Figure 9 is a running example of the *interval* method taking as input the same trajectory $\tilde{\mathcal{T}}[P_0, \dots, P_{10}]$. At the beginning, P_0 is the first start point, and points P_1, P_2, P_3, \dots , each has a *direction range* $Range(P_0P_1, \epsilon), Range(P_1P_2, \epsilon), Range(P_2P_3, \epsilon), \dots$, respectively. Because $\bigcap_{i=1}^4 Range(\overrightarrow{P_{0+i-1}P_{0+i}}, \epsilon) \neq \phi$ and $\overrightarrow{P_0P_4} \cdot \theta$ falls in the *common subinterval*, and $\bigcap_{i=1}^5 Range(\overrightarrow{P_{0+i-1}P_{0+i}}, \epsilon) = \phi$, $\overrightarrow{P_0P_4}$ is output and P_4 becomes the start point of the next section. At last, the algorithm outputs two continuous line segments $\overrightarrow{P_0P_4}$ and $\overrightarrow{P_4P_{10}}$. \square

5 TRAJECTORY AGING

Suppose that we have compressed a trajectory $\tilde{\mathcal{T}}_0$ to $\bar{\mathcal{T}}_1$ using any LS algorithm \mathcal{A} with error bound ϵ_1 . As time evolves, we may need to further compress $\bar{\mathcal{T}}_1$ to an even coarser trajectory $\bar{\mathcal{T}}_2$. What is the relationship among $\bar{\mathcal{T}}_2$, $\bar{\mathcal{T}}_1$ and $\tilde{\mathcal{T}}_0$? And what is the right way to get the coarser trajectory $\bar{\mathcal{T}}_2$? This section is to answer these questions from the views of friendliness [2] and errors. Note that if we get $\bar{\mathcal{T}}_1$ and $\bar{\mathcal{T}}_2$ by optimal algorithms w.r.t. error bounds ϵ_1 and ϵ_2 , respectively, then

Table 3. Aging friendliness and error bounds of line simplification algorithms for data aging

Algorithms	Distance Metrics	Aging Friendliness	Error Bounds
Optimal algorithms	PED or SED	✗ ([2])	$\epsilon_1 + \epsilon_2$ (Prop. 5.6)
	DAD	✗ (Prop. 5.1)	$\epsilon_1 + \epsilon_2$ (Prop. 5.6)
Batch algorithm DP	PED or SED	✓ ([2])	$\max(\epsilon_1, \epsilon_2)$ (Prop. 5.5)
	DAD	✗ (Prop. 5.2)	$\epsilon_1 + \epsilon_2$ (Prop. 5.6)
Batch algorithm TP	PED, SED or DAD	✗ (Prop. 5.3)	$\epsilon_1 + \epsilon_2$ (Prop. 5.6)
Online algorithms	PED, SED or DAD	✗ (Prop. 5.4)	$\epsilon_1 + \epsilon_2$ (Prop. 5.6)
One-pass algorithms	PED, SED or DAD	✗ (Prop. 5.4)	$\epsilon_1 + \epsilon_2$ (Prop. 5.6)



Fig. 10. A counter example of the aging friendliness of algorithm Optimal using DAD, where (1) the original trajectory $\{P_0, P_1, P_2, P_3\}$ is compressed using $\epsilon_2 = 45^\circ$ to three points $\{P_0, P_2, P_3\}$, and (2) it is first compressed using $\epsilon_1 = 30^\circ$ to three points $\{P_0, P_2, P_3\}$, then compressed using $\epsilon_2 = 45^\circ$ to two points $\{P_0, P_3\}$.

$\bar{\mathcal{T}}_2$ may not be the optimal one of $\ddot{\mathcal{T}}_0$ w.r.t. any error bound [2]. The major results are summarized in Table 3.

5.1 Friendliness of Data Aging

We first discuss the relationship between $\bar{\mathcal{T}}_1$ and $\bar{\mathcal{T}}_2$ from the view of friendliness, which was defined in [2], but was seldom discussed later.

Aging friendly [2]. An LS algorithm \mathcal{A} is aging friendly with respect to a distance metric \mathcal{M} if for every ϵ_1 and every ϵ_2 such that $\epsilon_1 < \epsilon_2$, and for every trajectory $\ddot{\mathcal{T}}$, $\mathcal{A}(\ddot{\mathcal{T}}, \epsilon_2, \mathcal{M}) = \mathcal{A}(\mathcal{A}(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$.

Cao et al. proved in [2] that “an optimal line simplification algorithm is not aging-friendly w.r.t. the PED and SED”, and “the top-down algorithm DP is aging friendly w.r.t. PED and SED” on the premise that the second run of DP takes as input the whole simplified trajectory produced by the first run. However, algorithms other than the optimal algorithms and the top-down algorithm DP with distance metrics other than PED and SED are not discussed in [2], thus, their effectiveness in data aging remains an open problem. In the rest, we present a full picture of this problem, starting from the optimal algorithms coupling with DAD.

PROPOSITION 5.1. *An optimal algorithm is not aging friendly w.r.t. DAD too.*

Proof: We shall prove this by constructing a counter-example shown in Figure 10, where error bounds $\epsilon_1 = 30^\circ$ and $\epsilon_2 = 45^\circ$.

(1) Optimal($\ddot{\mathcal{T}}, 45^\circ$, DAD). It first constructs the reachability graph of the trajectory that P_0 has arcs to P_1 and P_2 , P_1 has arcs to P_2 and P_3 , and P_2 has an arc to P_3 . At last, it outputs a shortest path of three points $\{P_0, P_2, P_3\}$.

(2) Optimal(Optimal($\ddot{\mathcal{T}}, 30^\circ$, DAD), 45° , DAD). In the first round ($\epsilon_1 = 30^\circ$), the original trajectory is compressed to three points $\{P_0, P_2, P_3\}$, and in the second round ($\epsilon_2 = 45^\circ$), because line segments $\overline{P_0P_2}$ and $\overline{P_2P_3}$ both have angular deviations to line segment $\overline{P_0P_3}$ less than 45° , it is finally compressed to two points $\{P_0, P_3\}$.

In this case, $\text{Optimal}(\ddot{\mathcal{T}}, 45^\circ, \text{DAD}) \neq \text{Optimal}(\text{Optimal}(\ddot{\mathcal{T}}, 30^\circ, \text{DAD}), 45^\circ, \text{DAD})$. Thus, algorithm Optimal is not aging friendly w.r.t. DAD. \square



Fig. 11. A counter example of the aging friendliness of DP using DAD, where (1) the original trajectory is compressed using $\epsilon_2 = 45^\circ$ to three points $\{P_0, P_2, P_4\}$, and (2) the original trajectory is first compressed using $\epsilon_1 = 30^\circ$ to three points $\{P_0, P_2, P_4\}$, then compressed using $\epsilon_2 = 45^\circ$ to two points $\{P_0, P_4\}$.



Fig. 12. A counter example of the aging friendliness of algorithm TP, where (1) the original trajectory is compressed using $\epsilon_2 = 6$ to three points $\{P_0, P_3, P_4\}$, and (2) the original trajectory is first compressed using $\epsilon_1 = 3$ to four points $\{P_0, P_1, P_3, P_4\}$, then compressed using $\epsilon_2 = 6$ to two points $\{P_0, P_4\}$.

PROPOSITION 5.2. *The top-down algorithm DP is not aging friendly w.r.t. DAD.*

Proof: We shall prove this by constructing a counter-example shown in Figure 11, where error bounds $\epsilon_1 = 30^\circ$ and $\epsilon_2 = 45^\circ$.

(1) DP($\ddot{\mathcal{T}}$, 45° , DAD). It finds that line segment $\overline{P_1P_2}$ has the largest angular deviation to line segment $\overline{P_0P_4}$ which is also larger than the error bound 45° , hence it uses point P_2 as the splitting point and splits the original trajectory to $\{P_0, P_1, P_2\}$ and $\{P_2, P_3, P_4\}$. At last, it outputs three points $\{P_0, P_2, P_4\}$.

(2) DP(DP($\ddot{\mathcal{T}}$, 30° , DAD), 45° , DAD). In the first round ($\epsilon_1 = 30^\circ$), the original trajectory is compressed to three points $\{P_0, P_2, P_4\}$, and in the second round ($\epsilon_2 = 45^\circ$), because line segments $\overline{P_0P_2}$ and $\overline{P_2P_4}$ both have angular deviations to line segment $\overline{P_0P_4}$ less than 45° , it is finally compressed to two points $\{P_0, P_4\}$.

In this case, $\text{DP}(\ddot{\mathcal{T}}, 45^\circ, \text{DAD}) \neq \text{DP}(\text{DP}(\ddot{\mathcal{T}}, 30^\circ, \text{DAD}), 45^\circ, \text{DAD})$. Thus, the DP algorithm is not aging friendly w.r.t. DAD. \square

The DP using DAD is different with PED and SED in that DAD is the deviation between two line segments rather than the deviation between a point and a line segment. For example, in Figure 11-(2), the angular deviations of line segments $\overline{P_1P_2}$ and $\overline{P_2P_4}$ to line segment $\overline{P_0P_4}$ are certainly different though they pass through the same point P_2 , thus, in the first round of run ($\epsilon_1 = 30^\circ$), the point P_2 serves as a splitting point while in the second round of run ($\epsilon_2 = 45^\circ$), it is no more a splitting point. This difference is the key that lets DP using DAD not aging friendly. We next discuss the aging friendliness of other algorithms.

PROPOSITION 5.3. *The bottom-up algorithm TP is not aging friendly w.r.t. PED, SED or DAD.*

Proof: We shall prove this by constructing a counter-example shown in Figure 12, where error bounds $\epsilon_1 = 3$ and $\epsilon_2 = 6$, and without losing generality, we use PED as the distance metric.

(1) TP($\ddot{\mathcal{T}}$, 6 , PED). It first merges $\overline{P_1P_2}$ and $\overline{P_2P_3}$ to $\overline{P_1P_3}$ as the merging of them has the lowest cost of 2, the distance from point P_2 to line segment $\overline{P_1P_3}$; then it merges $\overline{P_0P_1}$ and $\overline{P_1P_3}$ to $\overline{P_0P_3}$ with the lowest cost of 4.5, the distance from point P_1 to line segment $\overline{P_0P_3}$; at last, because the

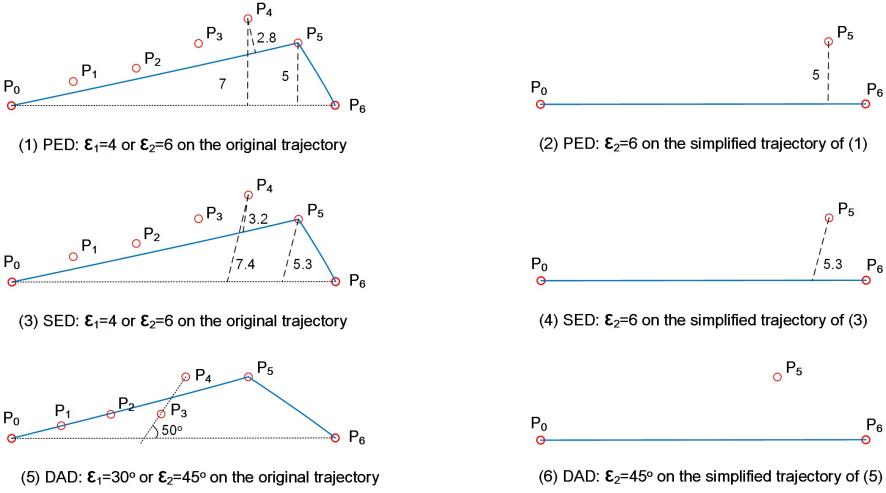


Fig. 13. Counter examples of the aging friendliness of incremental algorithms (either online or one-pass).

merging of $\overline{P_0P_3}$ and $\overline{P_3P_4}$ has a cost of 7, the distance from point P_2 to line segment $\overline{P_0P_4}$, which is larger than the error bound of 6, it outputs three points $\{P_0, P_3, P_4\}$.

(2) TP(TP($\tilde{\mathcal{T}}, 3$, PED), 6, PED). In the first round ($\epsilon_1 = 3$), the original trajectory is compressed to four points $\{P_0, P_1, P_3, P_4\}$, and in the second round ($\epsilon_2 = 6$), because all points in the result trajectory $\{P_0, P_1, P_3, P_4\}$ have distances to line segment $\overline{P_0P_4}$ less than 6, it is finally compressed to two points $\{P_0, P_4\}$.

In this case, $\text{TP}(\tilde{\mathcal{T}}, 6, \text{PED}) \neq \text{TP}(\text{TP}(\tilde{\mathcal{T}}, 3, \text{PED}), 6, \text{PED})$. Thus, TP is not aging friendly w.r.t. PED. Similarly, TP is not aging friendly w.r.t. SED or DAD. Hence, we have the conclusion. \square

PROPOSITION 5.4. *The online and one-pass algorithms are not aging friendly w.r.t. PED, SED or DAD.*

Proof: For online algorithm SQUISH-E, it supports SED only and runs in a bottom-up manner that is a slight variation of algorithm TP. As TP, it is not aging friendly w.r.t. SED. For other online and one-pass algorithms, though they apply different distance checking approaches, they run in a common incremental manner, i.e., they incrementally read data points until they can not represent those read points by one line segment, then they output the simplified sub-trajectory and continue to process the rest data points. We next construct counter examples to show that an incremental algorithm \mathcal{A} is not aging friendly.

(1) $\mathcal{A}(\tilde{\mathcal{T}}, \epsilon_2, \mathcal{M})$. As shown in Figure 13-(1)(3)(5), the algorithm \mathcal{A} incrementally reads $\{P_0, P_1, \dots, P_5\}$ and finds they can be represented by line segment $\overline{P_0P_4}$, thus the process progresses. Then, after point P_6 is read, it finds that these points can not be represented by any line segment, hence $\overline{P_0P_5}$ is output. Finally, the algorithm outputs $\{P_0, P_5, P_6\}$.

(2) $\mathcal{A}(\mathcal{A}(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$. When using $\epsilon_1 = 4$ or $\epsilon_1 = 30^\circ$, the algorithm also outputs $\{P_0, P_5, P_6\}$. Then $\{P_0, P_5, P_6\}$ is compressed using $\epsilon_2 = 6$ or $\epsilon_2 = 45^\circ$ to $\{P_0, P_6\}$, as shown in Figure 13-(2)(4)(6).

Combining (1) with (2), it is clear that the incremental algorithms are not aging friendly w.r.t. distance metric PED, SED or DAD. \square

5.2 Error of Data Aging

As most algorithms are not aging friendly, are they error bounded in data aging? if so, what are the bounds? This section focuses on these problems and discusses the error between the simplified trajectory $\tilde{\mathcal{T}}_2$ and the original trajectory $\tilde{\mathcal{T}}_0$.

PROPOSITION 5.5. *Given error bounds $\epsilon_1 > 0$ and $\epsilon_2 > 0$, for distance metric \mathcal{M} of PED and SED, the error bound between original trajectory $\tilde{\mathcal{T}}$ and simplified trajectory $\bar{\mathcal{T}} = DP(DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$ is $\max(\epsilon_1, \epsilon_2)$.*

Proof: We consider two cases: $\epsilon_2 \geq \epsilon_1$ and $\epsilon_2 < \epsilon_1$.

(1) For $\epsilon_2 \geq \epsilon_1$, as proved in [2], $DP(DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}) = DP(\tilde{\mathcal{T}}, \epsilon_2, \mathcal{M})$, which has the max error of ϵ_2 to the original trajectory $\tilde{\mathcal{T}}$.

(2) For $\epsilon_2 < \epsilon_1$, we shall prove $DP(DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}) = DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M})$ by induction on the number of points of $\tilde{\mathcal{T}}$.

- For a trajectory $\tilde{\mathcal{T}}$ with one or two points ($n = 1$ or $n = 2$), the simplified trajectories with any ϵ are sure identical to the original trajectory. Consider a trajectory $\tilde{\mathcal{T}} = [P_0, P_1, P_2]$ ($n = 3$), if the distance from P_1 to $\overline{P_0P_2}$ is less than ϵ_1 , then $DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M}) = [P_0, P_2]$. Obviously $DP(DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$ is $[P_0, P_2]$ too; if the distance from P_1 to $\overline{P_0P_2}$ is larger than ϵ_1 , then $DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M}) = \tilde{\mathcal{T}}$, and $DP(DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}) = DP(\tilde{\mathcal{T}}, \epsilon_2, \mathcal{M}) = \tilde{\mathcal{T}}$.
- Assume that it is true for every trajectory $\tilde{\mathcal{T}}$ having $n \geq 3$ points. Consider a trajectory with $n + 1$ points. Let d_{\max} denote the maximum distance between point P_i , $i \in [0, n]$, and the line segment $\overline{P_0P_n}$. If $d_{\max} < \epsilon_1$, then $DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M}) = [P_0, P_n]$, and $DP(DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}) = DP([P_0, P_n], \epsilon_2, \mathcal{M}) = [P_0, P_n]$. If $d_{\max} > \epsilon_1$, then in $DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M})$, point P_i will split the trajectory $\tilde{\mathcal{T}}$ into two sub-trajectories, i.e., $[P_0, \dots, P_i]$ and $[P_i, \dots, P_n]$, and continue to simplify each sub-trajectories. Hence, the result of $DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M})$ is the union of $DP([P_0, \dots, P_i], \epsilon_1, \mathcal{M})$ and $DP([P_i, \dots, P_n], \epsilon_1, \mathcal{M})$. Obviously, the points P_0 , P_i and P_n are in the simplified trajectory of $DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M})$, and P_i is still the first splitting point of the DP algorithm taking the simplified trajectory and ϵ_2 as input. Hence, $DP(DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$ is the union of $DP(DP([P_0, \dots, P_i], \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$ and $DP(DP([P_i, \dots, P_n], \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$. By the assumption, we have $DP(DP([P_0, \dots, P_i], \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}) = DP([P_0, \dots, P_i], \epsilon_1, \mathcal{M})$ and $DP(DP([P_i, \dots, P_n], \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}) = DP([P_i, \dots, P_n], \epsilon_1, \mathcal{M})$. Thus, $DP(DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}) = DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M})$.
- Now we have $DP(DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}) = DP(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M})$, whose max error to the original trajectory $\tilde{\mathcal{T}}$ is ϵ_1 .

Combining (1) with (2), we have the conclusion. \square

PROPOSITION 5.6. *Given error bounds $\epsilon_1 > 0$ and $\epsilon_2 > 0$, for any LS algorithm \mathcal{A} and distance metric \mathcal{M} of PED, SED and DAD other than DP using PED and SED, the error bound between original trajectory $\tilde{\mathcal{T}}$ and simplified trajectory $\bar{\mathcal{T}} = \mathcal{A}(\mathcal{A}(\tilde{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$ is $\epsilon_1 + \epsilon_2$.*

Proof: We shall prove this by showing that the error bound is neither more than $\epsilon_1 + \epsilon_2$ nor less than $\epsilon_1 + \epsilon_2$, from which we have that the error bound is exactly $\epsilon_1 + \epsilon_2$.

(1) We first prove that the error bound is not more than $\epsilon_1 + \epsilon_2$. Suppose that a point P_k is represented by line segment $\overline{P_iP_j}$ with error bound ϵ_1 , and points P_i and P_j are further represented by line segment $\overline{P_sP_t}$ with error bound ϵ_2 (Figure 14). If the distance metric is PED, then the distance from P'_k to $\overline{P_sP_t}$ is less than ϵ_2 , hence, the distance from P_k to $\overline{P_sP_t}$ is less than $\epsilon_1 + \epsilon_2$. If it is SED, then $|P_iP''_i| < \epsilon_2$, $|P_jP''_j| < \epsilon_2$, and $\frac{|P'_iP''_k|}{|P'_kP'_j|} = \frac{|P''_iP''_k|}{|P''_kP''_j|}$, hence, $|P'_kP''_k| < \epsilon_2$, and the distance from P_k

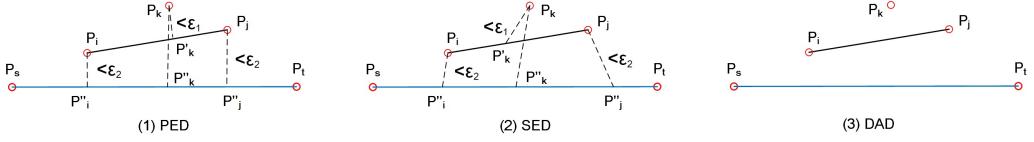


Fig. 14. Examples of aging errors.

to $\overline{P_s P_t}$, i.e., $|P_k P''_k|$, is less than $\epsilon_1 + \epsilon_2$. If it is DAD, then obviously the error between $\overline{P_k P_{k+1}}$ and $\overline{P_s P_t}$ is not more than $\epsilon_1 + \epsilon_2$.

(2) We then prove that the error bound is not less than $\epsilon_1 + \epsilon_2$. If \mathcal{A} is a top-down algorithm using DAD, then from Figures 11-(2) we can find that the error from $\overline{P_3 P_4}$ to $\overline{P_0 P_4}$ is $\angle P_3 P_4 P_0 = \angle P_3 P_4 P_2 + \angle P_2 P_4 P_0$, whose bound is not less than $\epsilon_1 + \epsilon_2$, thus the error bound between original trajectory \mathcal{T} and simplified trajectory $\overline{\mathcal{T}} = \mathcal{A}(\mathcal{A}(\mathcal{T}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$ is not less than $\epsilon_1 + \epsilon_2$. If \mathcal{A} is a bottom-up algorithm or an incremental algorithm, either online or one-pass, then from Figure 12-(2) or Figure 13 we also have the conclusion.

Combining (1) with (2), we have the conclusion. \square

6 EVALUATION

In this section, we present extensive and systematic experimental studies and analyses of eleven representative LS algorithms. Using three real-life datasets, we conduct five sets of tests to evaluate the effectiveness in terms of compression ratios and errors, efficiency (running time), aging friendliness and query friendliness of these representative algorithms using distance metrics PED, SED and DAD, and the impacts of error bounds ϵ and trajectory sizes.

6.1 Experimental Setting

Real-life Trajectory Datasets. We use three real-life datasets shown in Table 4, namely, Service car trajectory data (UCar), Geolife trajectory data (Geolife) [43] and Mopsi trajectory data (Mopsi) [19], to evaluate those LS algorithms. These data sets come from different sources, where UCar is collected by cars in urban, and Geolife and Mopsi are a mixing of cars and individuals. They also have typical sampling rates used in practice, ranging from one point per second to one point per five seconds. The data source and sampling rate also affect the performance of LS algorithms using certain distance metrics.

- (1) UCar is the GPS trajectories collected by a Chinese car rental company during Apr. 2015 to Nov. 2015. Most routes are located in big cities. The sampling rate is one point per 3–5 seconds, and each trajectory has around 114.1K points.
- (2) Geolife is the GPS trajectories collected in the GeoLife project [43] by 182 users in a period from Apr. 2007 to Oct. 2011. These trajectories have a variety of sampling rates, among which 91% are logged at one point per 1–5 seconds. The longest trajectory has 2,156,994 points.
- (3) Mopsi is the GPS trajectories collected in the Mopsi project [19] by 51 users in a period from 2008 to 2014. Most routes are located in Joensuu region, Finland. The sampling rate is one point per 2 seconds, and each trajectory has around 153.9K points.

Algorithms and implementation. We have implemented the representative algorithms shown in Table 1. They are optimal algorithm Optimal, batch algorithms DP and TP, online algorithms OPW, BQS and SQUISH-E, and one-pass algorithms OPERB, SIPED, CISED, Intersect and Interval. For one-pass algorithms SIPED and CISED, we implement two versions of them (half and full ϵ), denoted as SIPED (ϵ), SIPED ($\frac{\epsilon}{2}$), CISED (ϵ) and CISED ($\frac{\epsilon}{2}$). For algorithm CISED (ϵ) and CISED

Table 4. Real-life trajectory datasets

Data sets	Number of Trajectories	Sampling Rates (s)	Points Per Trajectory	Total Points
UCar	200	3-5	~ 114.0K	22.8M
Geolife [43]	182	1-5	~ 131.4K	24.2M
Mopsi [19]	51	2	~ 153.9K	7.9M

($\frac{\epsilon}{2}$), we fixed parameter $m = 16$ as evaluated in [14], *i.e.*, 16-edges inscribe regular polygon. For algorithm OPERB, we remove its fifth optimization technique to make it fit for the definition of error bounded algorithms. All algorithms are implemented with Java. All tests are run on an x64-based PC with 4 Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz and 8GB of memory, and the max heap size of Java VM is 4GB.

We test these algorithms under varied error bounds ϵ and trajectory sizes, respectively. We first vary ϵ from $10m$ to $100m$ in PED and SED (or from 15° to 90° in DAD) on the entire four datasets, respectively. We then choose 10 trajectories from each dataset, and vary the size $|\ddot{\mathcal{T}}|$ of a trajectory from 1,000 points to 10,000 points while fixing the error bound $\epsilon = 40$ metres or $\epsilon = 45$ degrees.

6.2 Evaluation Metrics

Compression ratios and errors are the most popular metrics to evaluate the effectiveness of LS algorithms, and they are also the metrics to evaluate the aging friendliness and query friendliness of LS algorithms. And the running time is used to evaluate the efficiency of LS algorithms.

Compression ratios. For trajectories $\{\ddot{\mathcal{T}}_1, \dots, \ddot{\mathcal{T}}_M\}$ and their piece-wise line representations $\{\overline{\mathcal{T}}_1, \dots, \overline{\mathcal{T}}_M\}$, the compression ratio is $(\sum_{j=1}^M |\overline{\mathcal{T}}_j|)/(\sum_{j=1}^M |\ddot{\mathcal{T}}_j|)$. By this definition, algorithms with lower compression ratios are better.

Average Errors. All these algorithms in Table 1 are error bounded, *i.e.*, the max errors are bounded. Hence, we only evaluate the average errors here. Given a set of trajectories $\{\ddot{\mathcal{T}}_1, \dots, \ddot{\mathcal{T}}_M\}$ and their piece-wise line representations $\{\overline{\mathcal{T}}_1, \dots, \overline{\mathcal{T}}_M\}$, and $P_{j,i}$ denoting a point in trajectory $\ddot{\mathcal{T}}_j$ contained in a line segment $\mathcal{L}_{l,i} \in \overline{\mathcal{T}}_l$ ($l \in [1, M]$), then the average error is $\sum_{j=1}^M \sum_{i=0}^M d(P_{j,i}, \mathcal{L}_{l,i})/\sum_{j=1}^M |\ddot{\mathcal{T}}_j|$.

Running time. It is the efficiency of algorithms.

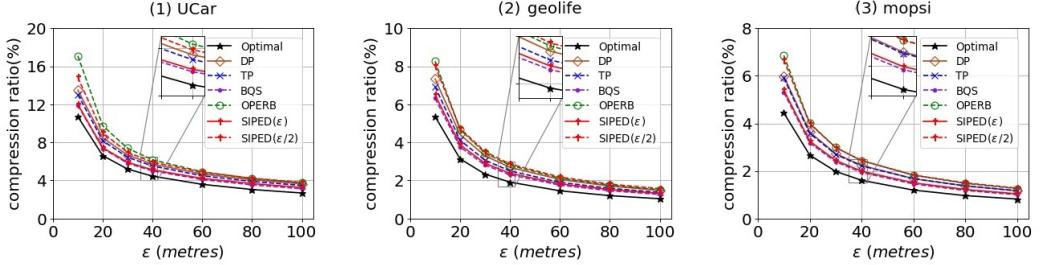
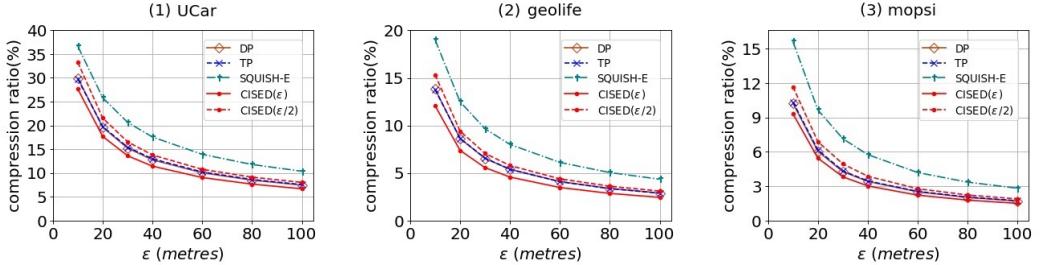
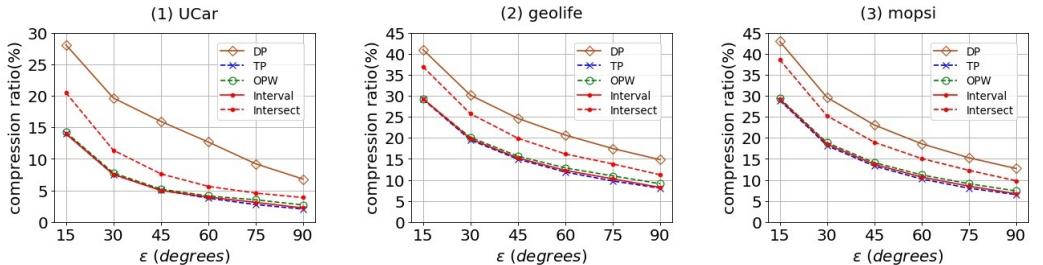
6.3 Experimental Results and Analyses

We next present our findings.

6.3.1 Evaluation and Analyses of Compression Ratio. The compression ratios of these algorithms under varied error bounds ϵ and trajectory sizes are reported in Figures 15, 16, 17, 18, 19 and 20. Note that the optimal algorithm using SED and DAD is not reported in Figures 15, 16 and 17 as it runs out of memory when compressing the full dataset. We first report our findings.

(1) Datasets may have impacts on the compression ratios of LS algorithms. Datasets with higher sampling rates typically have better compression ratios for PED and SED, while it is opposite for DAD. The dataset collected by cars (*e.g.*, UCar) also has better compression ratios than the datasets partially collected by individuals (*e.g.*, Geolife and Mopsi), as cars typically move more regularly than individuals.

(2) The compression ratios of algorithms using PED from the best to the worst are the optimal algorithm Optimal, online algorithm BQS, one-pass algorithm SIPED (ϵ), batch algorithms TP and DP, and one-pass algorithms SIPED ($\frac{\epsilon}{2}$) and OPERB. The output sizes of algorithms BQS and SIPED (ϵ) are on average (113.32%, 120.22%, 120.83%) and (116.04%, 124.46%, 124.24%) of algorithm Optimal on datasets (UCar, Geolife, Mopsi), respectively. Algorithms TP and DP are comparable,

Fig. 15. Evaluation of compression ratios (PED) on full datasets: varying the error bound ϵ .Fig. 16. Evaluation of compression ratios (SED) on full datasets: varying the error bound ϵ .Fig. 17. Evaluation of compression ratios (DAD) on full datasets: varying the error bound ϵ .

and their output sizes are on average (125.05%, 131.01%, 138.01%) and (130.03%, 140.56%, 139.00%) of Optimal on datasets (UCar, Geolife, Mopsi), respectively. Algorithms SIPED ($\frac{\epsilon}{2}$) and OPERB are comparable, and they are on average (136.73%, 150.23%, 152.29%) and (143.14%, 147.80%, 152.37%) of Optimal on datasets (UCar, Geolife, Mopsi), respectively. For example, in Mopsi, the compression ratios of algorithms (Optimal, TP, DP, BQS, SIPED (ϵ), SIPED ($\frac{\epsilon}{2}$), OPERB) are (1.6%, 2.2%, 2.2%, 1.9%, 2.0%, 2.4%, 2.4%) when $\epsilon = 40m$.

(3) The compression ratios of algorithms using SED from the best to the worst are the Optimal algorithm, one-pass algorithm CISED (ϵ), batch algorithms TP and DP, one-pass algorithm CISED ($\frac{\epsilon}{2}$), and online algorithm SQUISH-E. Algorithms TP and DP are comparable, and they are on average (125.23%, 143.92%, 128.63%) and (123.93%, 141.46%, 121.14%) of algorithm Optimal on datasets (UCar, Geolife, Mopsi), respectively. Algorithms CISED (ϵ), CISED ($\frac{\epsilon}{2}$) and SQUISH-E are on average (109.27%, 110.13%, 115.90%), (134.35%, 159.30%, 136.06%) and (165.94%, 225.68%, 206.90%) of Optimal on (UCar, Geolife, Mopsi), respectively. For example, in Mopsi, the compression ratios of algorithms (TP, DP, SQUISH-E, CISED (ϵ), CISED ($\frac{\epsilon}{2}$))) are (3.45%, 3.41%, 5.75%, 3.02%, 3.86%), respectively, when $\epsilon = 40m$.

(4) The compression ratios of algorithms using DAD from the best to the worst are the Optimal algorithm, batch algorithm TP and one-pass algorithm Interval, online algorithm OPW, one-pass

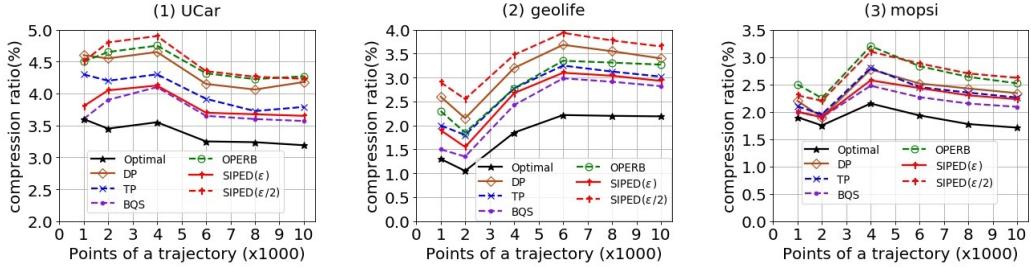


Fig. 18. Evaluation of compression ratios (PED) on small datasets: varying the size of trajectories.

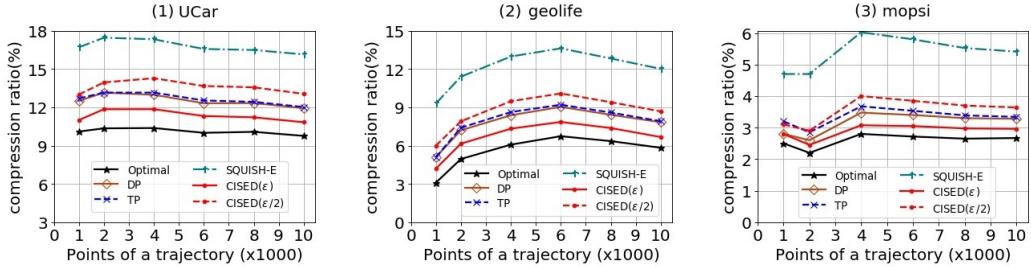


Fig. 19. Evaluation of compression ratios (SED) on small datasets: varying the size of trajectories.

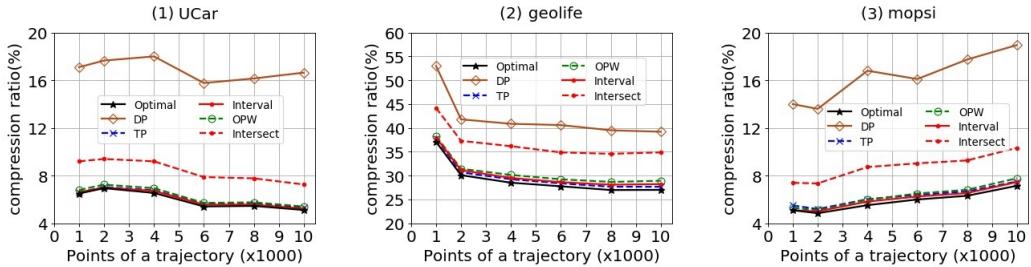


Fig. 20. Evaluation of compression ratios (DAD) on small datasets: varying the size of trajectories.

algorithm Intersect and batch algorithm DP. Algorithms TP, OPW and Interval are comparable, and are on average (102.91%, 102.27%, 106.88%), (116.09%, 107.11%, 115.42%) and (101.98%, 103.52%, 103.43%) of algorithm Optimal on datasets (UCar, Geolife, Mopsi), respectively. Algorithms Intersect and DP are on average (156.00%, 121.20%, 230.52%) and (283.93%, 143.79%, 278.89%) of algorithm Optimal on datasets (UCar, Geolife, Mopsi), respectively. For example, in Mopsi, the compression ratios of algorithms (TP, DP, OPW, Interval, Intersect) are (13.3%, 23.1%, 14.12%, 13.7%, 18.96%), respectively, when $\epsilon = 45$ degrees.

We then present analyses from the views of LS algorithms and distance metrics.

Analyses of LS algorithms. The Optimal algorithm is the best in terms of compression ratios, followed by online algorithms OPW and BQS and one-pass algorithms using the full ϵ sector/cone/range. One-pass algorithms using a half ϵ sector/cone/range and batch algorithms except DP using DAD also have good compression ratios.

For batch algorithms, bottom-up algorithm (TP) and top-down algorithm (DP) have the similar compression ratios when using PED and SED. However, when using DAD, bottom-up methods have obviously better compression ratios than top-down methods. As we know that top-down algorithms split a long trajectory $[P_s, \dots, P_e]$ into two sub trajectories by finding out a splitting point $P_i (s < i < e)$ that has the max position deviation (or whose line segment $\overrightarrow{P_{i-1}P_i}$ has the max

direction deviation) to line segment $\overrightarrow{P_s P_e}$. Though this strategy works well with PED and SED, a point with the max direction deviation may not be a reasonable splitting point in the direction-aware scenario. Thus it leads to a poorer compression ratio. However, bottom-up methods do not have this weakness as they always merge neighbouring points.

For online algorithms, BQS and OPW are comparable with the best sub-optimal algorithms. This is because OPW is indeed a combination of DP and opening window, and BQS is mainly an efficiency optimized OPW. SQUISH-E has the poorest compression ratio among all algorithms using SED. This is the result of its mechanism: SQUISH-E estimates the lowest SED error and removes the point with “predicted to introduce the lowest amount of error into the compression”[24]. Its “prediction” method is not accurate enough, thus, in order to ensure the error bound, it may ignore too many potential points that could be represented by a line segment.

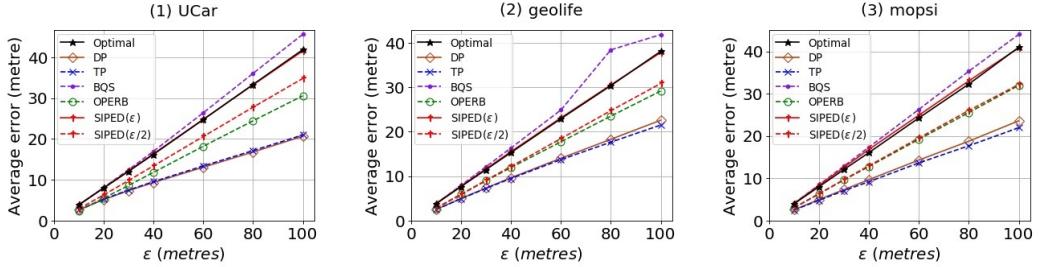
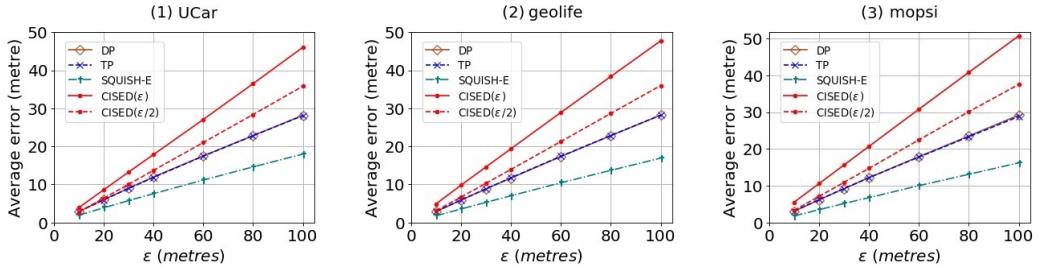
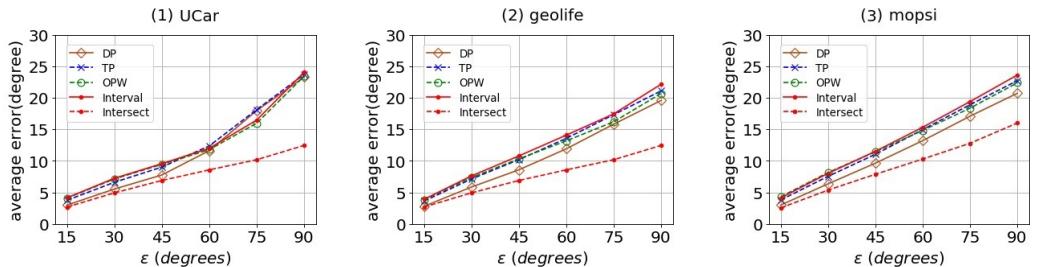
For one-pass algorithms, the full ϵ sector/cone/range combining with a position/direction constraint always has better compression ratios than the half ϵ sector/cone/range versions in all datasets, and they are comparable with the best sub-optimal algorithms. This may be related to the moving habits or patterns of moving objects that implied in trajectories. That is, a moving object, like an individual or a car, usually keeps moving forward for quite a long time, engendering a sequence of data points distributing in a narrow strip. Under such circumstance, a new data point is quite possibly living in the common intersection of larger sectors/cones/ranges, which further leads to a better compression ratio.

Analyses of distance metrics. Though PED, SED and DAD are different distances, the comparison of their compression ratios is helpful to choose an effective distance metric. First, given the same error bound ϵ , the compression ratios of algorithms using PED are obviously better than using SED. More specifically, *the output sizes of using SED are approximately twice of PED*. As shown in Figures 15 and 16, the output sizes of algorithms TP and DP using PED are on average (43.55%, 47.49%, 63.15%) and (45.79%, 50.88%, 64.50%) of algorithms TP and DP using SED on datasets (UCar, Geolife, Mopsi), respectively. This result shows SED saves temporal information at a price of twice more data points.

Secondly, in practice (*e.g.*, $\epsilon < 100$ meters and $\epsilon < 60$ degrees), SED have obviously better compression ratios than DAD in datasets Geolife and Mopsi and a bit poorer than DAD in UCar. This is because some Geolife and Mopsi trajectories are collected by individuals that are in transportation modes of walking, running and riding, and moving objects in those modes may change their directions with a considerable range (*e.g.*, large than 60 degrees) more frequently than cars in urban. Moreover, Geolife and Mopsi have higher sampling rates than UCar, which capture more direction changes, *i.e.*, direction changes in a small time interval.

6.3.2 Evaluation and Analyses of Average Error. The average errors of these algorithms, under varied error bounds ϵ and trajectory sizes, are reported in Figures 21, 22, 23, 24, 25 and 26. We first report our findings.

- (1) Datasets are insensitive to the errors of LS algorithms.
- (2) When using PED, the average errors from the smallest to the largest are batch algorithms TP and DP, one-pass algorithms SIPED ($\frac{\epsilon}{2}$) and OPERB, the optimal algorithm Optimal and one-pass algorithm SIPED (ϵ), and online algorithm BQS. For full datasets, algorithms TP and DP are comparable, and they are on average (58.69%, 61.34%, 57.57%) and (57.61%, 62.66%, 60.23%) of Optimal on datasets (UCar, Geolife, Mopsi), respectively. Algorithms SIPED ($\frac{\epsilon}{2}$) and OPERB are comparable, and they are on average (80.96%, 79.12%, 79.33%) and (70.60%, 76.64%, 78.71%) of Optimal on datasets (UCar, Geolife, Mopsi), respectively. Algorithms SIPED (ϵ) and BQS are on average (100.05%, 101.01%, 102.69%) and (104.67%, 108.91%, 106.92%) of Optimal on datasets (UCar,

Fig. 21. Evaluation of average errors (PED) on full datasets: varying the error bound ϵ .Fig. 22. Evaluation of average errors (SED) on full datasets: varying the error bound ϵ .Fig. 23. Evaluation of average errors (DAD) on full datasets: varying the error bound ϵ .

Geolife, Mopsi), respectively. For example, the average errors of algorithms (Optimal, TP, DP, BQS, SIPED (ϵ), SIPED ($\frac{\epsilon}{2}$), OPERB) in the full Mopsi are (16.08, 9.19, 9.68, 17.4, 12.96, 16.83, 12.77) metres when $\epsilon = 40m$.

(3) When using SED, the average errors from the smallest to the largest are online algorithm SQUISH-E, batch algorithms TP and DP, one-pass algorithm CISED ($\frac{\epsilon}{2}$), and one-pass algorithm CISED (ϵ) and the Optimal algorithm. Algorithms TP and DP are comparable, and they are on average (60.36%, 66.11%, 62.43%) and (62.54%, 67.04%, 68.64%) of Optimal on datasets (UCar, Geolife, Mopsi), respectively. Algorithms CISED (ϵ), CISED ($\frac{\epsilon}{2}$) and SQUISH-E are on average (97.32%, 106.74%, 108.16%), (75.29%, 76.03%, 81.44%) and (40.61%, 38.15%, 34.22%) of Optimal on datasets (UCar, Geolife, Mopsi), respectively. For example, the average errors of algorithms (Optimal, TP, DP, SQUISH-E, CISED (ϵ), CISED ($\frac{\epsilon}{2}$)) in full Mopsi are (19.39, 12.17, 12.20, 6.76, 20.68, 14.71) metres, respectively, when $\epsilon = 40m$.

(4) When using DAD, the average errors from the smallest to the largest are one-pass algorithm Intersect, batch algorithms DP and TP, one-pass algorithm Interval and online algorithm OPW, and the optimal algorithm Optimal. Algorithms TP, OPW and Interval are comparable, and they are on average (91.35%, 61.45%, 73.71%), (91.95%, 61.37%, 76.17%) and (90.36%, 68.23%, 163.47%) of Optimal on datasets (UCar, Geolife, Mopsi), respectively. Algorithms Intersect and DP are on

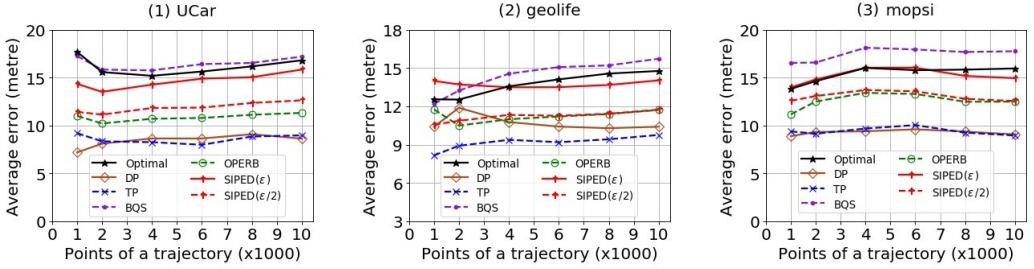


Fig. 24. Evaluation of average errors (PED) on small datasets: varying the size of trajectories.

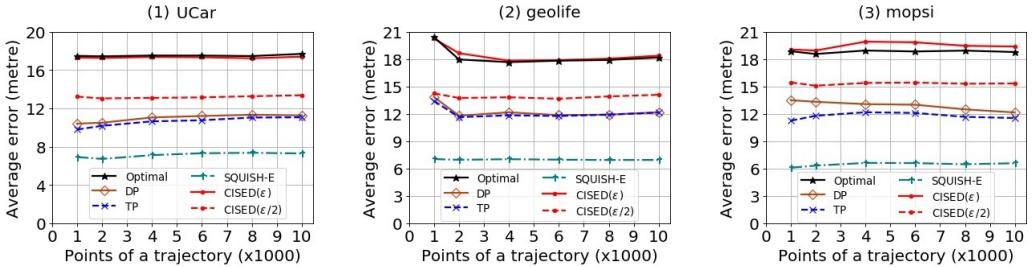


Fig. 25. Evaluation of average errors (SED) on small datasets: varying the size of trajectories.

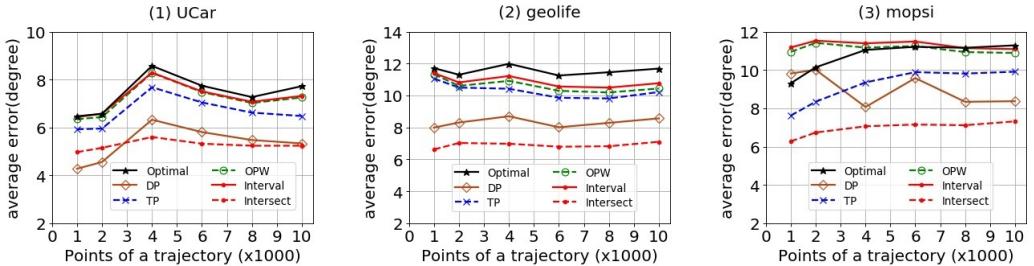


Fig. 26. Evaluation of average errors (DAD) on small datasets: varying the size of trajectories.

average (62.03%, 76.54%, 110.69%) and (82.45%, 96.52%, 137.95%) of Optimal on datasets (UCar, Geolife, Mopsi), respectively.

We then present analyses from the views of LS algorithms and distance metrics.

Analyses of LS algorithms. The average errors of these algorithms are generally on the contrary of compression ratios. The optimal algorithm is usually the worst algorithm in terms of average errors, followed by one-pass algorithms and then batch algorithms. Online algorithms have varied average errors, ranging from the best to the worst. (1) For batch algorithms, both bottom-up algorithm (TP) and top-down algorithm (DP) have similar average errors, and they are pretty good compared with other algorithms. (2) Online algorithms BQS and OPW often have the largest average errors in all sub-optimal algorithms, while SQUISH-E has the smallest. This is also on the contrary of their compression ratios. (3) For one-pass algorithms, the full ϵ sector/cone/range combining with a position/direction constraint always have larger average errors than the half ϵ sector/cone/range. Local distance checking approaches try to include more points into a line segment, this greedy strategy is likely leading to larger average errors, considerable larger than batch algorithms that have the similar compression ratios as one-pass and online algorithms.

Analyses of distance metrics. For the same error bound ϵ , the average errors of algorithms using SED are a bit larger than using PED. As we know that PED error is originally caused by the

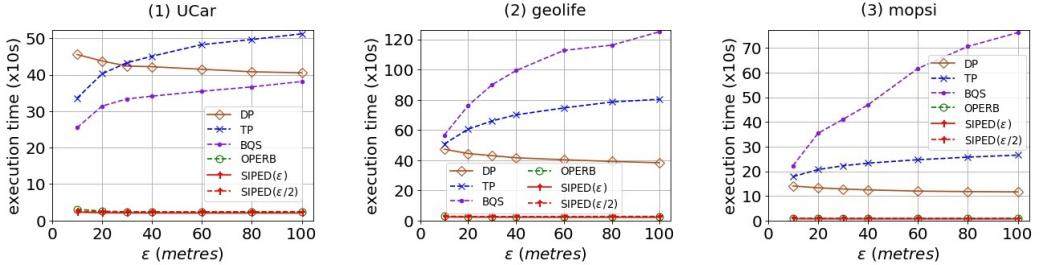


Fig. 27. Evaluation of running time (PED) on full datasets: varying the error bound ϵ .

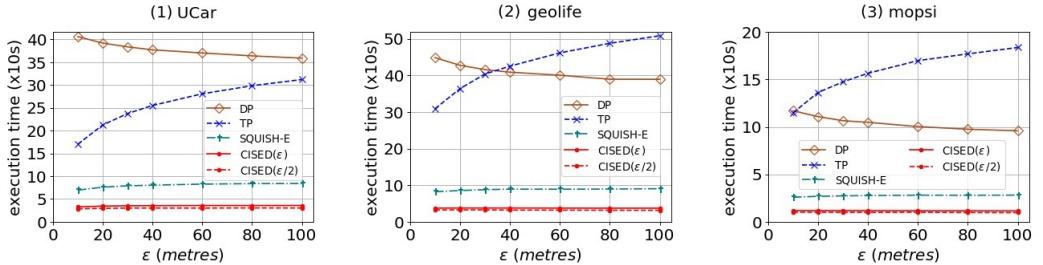


Fig. 28. Evaluation of running time (SED) on full datasets: varying the error bound ϵ .

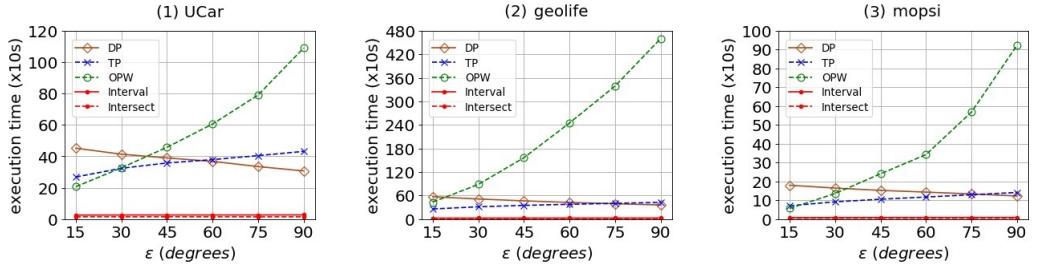


Fig. 29. Evaluation of running time (DAD) on full datasets: varying the error bound ϵ .

direction changes of a moving object while SED error is caused by the changes of both the direction and the speed of a moving object, the above phenomenon probably reveals that the changes of speeds are more frequent than the changes of directions for moving objects. And in practice (e.g., $\epsilon = 60$ meters and $\epsilon = 45$ degrees), the average errors of algorithms using DAD, when translated to position errors like PED, are likely 10 times larger than algorithms directly using PED and SED. This is obvious as a small direction deviation with a long trip may lead to a large position error.

6.3.3 Evaluation and Analyses of Efficiency. In this set of tests, we compare the efficiency of these algorithms. The results are reported in Figures 27, 28, 29, 30, 31 and 32. Note that even on the small datasets, *the running time of algorithm Optimal is thousands of times slower than one-pass algorithms*. As it is not clear to show all these algorithms in a single figure, only the results of sub-optimal algorithms are shown in these figures. We first report our findings.

- (1) Datasets do not have obvious impacts on the running time of LS algorithms.
- (2) When using PED, in most cases, the running time from the smallest to the largest is one-pass algorithms SIPED and OPERB, batch algorithms TP and DP, and online algorithm BQS. Algorithms SIPED ($\frac{\epsilon}{2}$) and OPERB are comparable, algorithm SIPED (ϵ) is $(0.92, 0.92, 0.91)$ times of SIPED ($\frac{\epsilon}{2}$), and algorithms TP, DP and BQS are on average $(26.79, 28.25, 29.87)$, $(16.32, 15.40, 11.02)$ and $(37.73, 62.23, 61.29)$ times slower than one-pass algorithm SIPED ($\frac{\epsilon}{2}$) on datasets (UCar, Geolife,

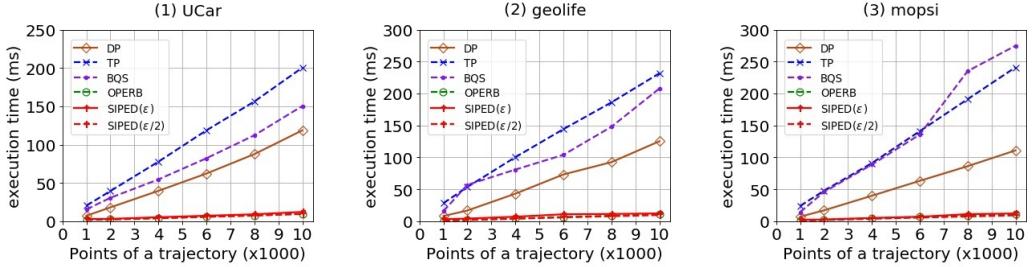


Fig. 30. Evaluation of running time (PED) on small datasets: varying the size of trajectories.

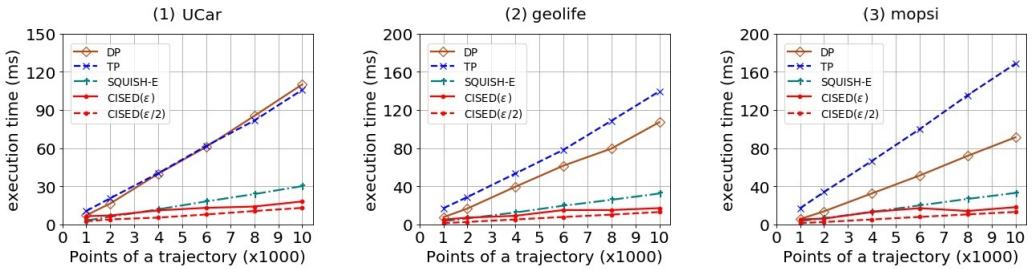


Fig. 31. Evaluation of running time (SED) on small datasets: varying the size of trajectories.

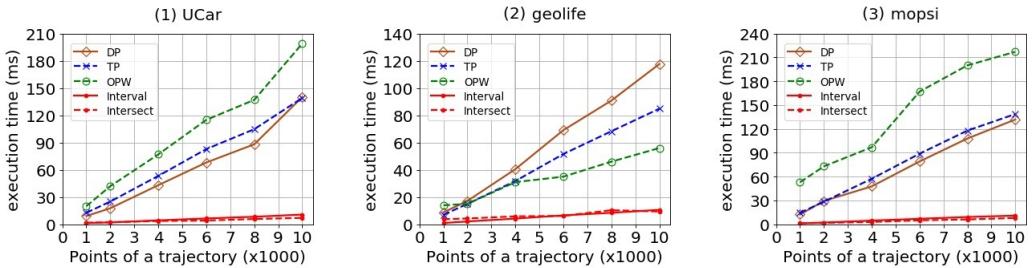


Fig. 32. Evaluation of running time (DAD) on small datasets: varying the size of trajectories.

Mopsi), respectively. For example, in Mopsi, the running time of algorithms (TP, DP, BQS, SIPED (ϵ), SIPED ($\epsilon/2$), OPERB) is (232.9, 124.2, 469.4, 6.89, 7.6, 8.6) seconds when $\epsilon = 40m$.

- (3) When using SED, the running time from the smallest to the largest is one-pass algorithm CISED, online algorithm SQUISH-E, and batch algorithms TP and DP. Algorithm CISED (ϵ) is (1.17, 1.17, 0.91) times of CISED ($\epsilon/2$), and algorithms TP, DP and SQUISH-E are on average (13.33, 15.81, 13.09), (12.93, 10.64, 8.79) and (2.75, 2.78, 2.57) times slower than CISED ($\epsilon/2$) on datasets (UCar, Geolife, Mopsi), respectively. For example, in Mopsi, the running time of algorithms (TP, DP, SQUISH-E, CISED (ϵ), CISED ($\epsilon/2$)) is (156.6, 104.8, 27.2, 11.6, 9.7) seconds when $\epsilon = 40m$.
- (4) When using DAD, the running time from the smallest to the largest is one-pass algorithms Intersect and Interval, batch algorithms TP and DP, and online algorithm OPW. Algorithm Interval is (1.80, 1.84, 1.81) times slower than Intersect, and algorithms TP, DP and OPW are on average (24.63, 23.53, 23.23), (25.49, 30.11, 31.72) and (39.29, 147.85, 80.09) times slower than Intersect on datasets (UCar, Geolife, Mopsi), respectively. For example, the running time of algorithms (TP, DP, OPW, Interval, Intersect) is (105.57, 152.53, 240.40, 8.57, 4.69) seconds in Mopsi when $\epsilon = 45$ degrees, respectively.

We then present analyses from the views of LS algorithms and distance metrics.

Analyses of LS algorithms. The running time from the fastest to the slowest is one-pass algorithms, online and batch algorithms, and optimal algorithms.

For batch algorithms, the running time of DP and TP decreases or increases with the increase of error bound ϵ , respectively, due to the top-down and bottom-up approaches that they apply. When using PED or SED, top-down algorithm usually runs faster than bottom-up algorithm when the error bound ϵ is large (e.g., in Geolife, $\epsilon > 10$ metres when using PED and $\epsilon > 30$ metres when using SED), which means that top-down (bottom-up) algorithm needs to split (merge) the original trajectory fewer (more) times in these cases, vice versa. When using DAD, top-down algorithms are normally a bit slower than bottom-up algorithms (recall that top-down algorithms have poorer compression ratios compared with bottom-up algorithms, which means that it needs more time to split the raw trajectory into more sub trajectories). In addition to error bounds, sampling rates also have impacts on the efficiency of batch algorithms. A dataset with high sampling rate likely needs more merging processes than splitting processes, thus, top-down algorithms run faster than bottom-up algorithms in high sampling datasets when using PED or SED.

For online algorithms, SQUISH-E is faster than BQS and OPW at a cost of poorer compression ratios, and it is still a few times slower than one-pass algorithms. BQS and OPW both have poor efficiency as they finally need batch approaches to simplify buffered data, and batch approaches running in a buffer are still time consuming.

For one-pass algorithms, OPERB, SIPED, CISED and Interval show a linear running time that is consistent with their time complexity analyses. They are not very sensitive to error bound ϵ , and also scale well with the increase of trajectory size on all datasets as a data point is processed only one time during the whole process. Algorithms SIPED, OPERB and Interval have similar running time, and algorithm CISED runs a bit slower than them, partially because finding the common intersection of spatial-temporal cones is a heavier work than sectors or ranges.

Analyses of distance metrics. The computation time of DAD is faster than PED and SED, and the computation time of PED and SED are 2.3 and 1.7 times of DAD, respectively. It is also worth pointing out that algorithms DP using PED, SED and DAD have similar running time in all datasets, though the computation of PED is much heavier than SED and DAD. The reason is that DP using PED has the best compression ratios which instead leads to the least splitting processes in the top-down manner. Combining these two factors, i.e., the computing of distance/direction deviation and the processing of trajectory splitting, finally, algorithm DP using PED has similar running time as DP using DAD or SED.

6.3.4 Evaluation and Analyses of Aging Friendliness. In this set of tests, we compare the errors and compression ratios of algorithms in data aging. We set $\epsilon_1 = 40m$ (or $\epsilon_1 = 30^\circ$) in the first run and $\epsilon_2 = 60m$ (or $\epsilon_2 = 50^\circ$) in the second run. Besides, we also run these algorithms on the raw trajectories, setting $\epsilon_3 = \epsilon_1 + \epsilon_2 = 100m$ (or $\epsilon_3 = 80^\circ$). The max errors are reported in [Table 5](#), and the compression ratios are reported in [Table 6](#) and [Table 7](#), respectively.

- (1) Algorithms DP using PED and SED both have max errors less than $\epsilon_2 = 60m$, confirming that they are aging friendly and their max errors are consistent with Theorem 5.5; while algorithm DP using DAD and other algorithms have max error larger than $\epsilon_2 = 60m$ or $\epsilon_2 = 50^\circ$, confirming that they are not aging friendly and their max errors are consistent with Theorems 5.2, 5.3 and 5.4.
- (2) Algorithm DP using DAD and other algorithms have max error less than $\epsilon_1 + \epsilon_2 = 100m$ or $\epsilon_1 + \epsilon_2 = 80^\circ$, which is consistent with Theorem 5.6.
- (3) [Table 6](#) and [Table 7](#) tell that, if algorithms compress data using ϵ_1 and ϵ_2 in turn, then they have a bit poorer compression ratios than directly using $\epsilon_3 = \epsilon_1 + \epsilon_2$. Note that the causes of this phenomenon are varied. For algorithm DP using PED or SED, it is caused by the shorten of the

Table 5. The max errors of algorithms in data aging that set $\epsilon_1 = 40m$ (or $\epsilon_1 = 30^\circ$ when using DAD) in the first run and $\epsilon_2 = 60m$ (or $\epsilon_2 = 50^\circ$ when using DAD) in the second run.

Alg. (PED)	UCar	Geolife	Mopsi	Alg. (SED)	UCar	Geolife	Mopsi	Alg. (DAD)	UCar	Geolife	Mopsi
DP	59.99	59.99	59.99	DP	59.99	59.99	59.99	DP	79.90	79.93	78.96
TP	99.56	96.95	93.00	TP	97.55	96.70	90.53	TP	79.94	79.93	79.64
BQS	96.20	93.58	96.79	SQUISH-E	92.60	90.89	84.38	OPW	79.96	79.96	79.74
SIPED (ϵ)	97.34	95.21	98.86	CISED (ϵ)	97.75	97.51	96.65	Interval	79.96	79.93	79.74
SIPED ($\frac{\epsilon}{2}$)	99.18	94.57	97.91	CISED ($\frac{\epsilon}{2}$)	97.40	97.51	98.83	Intersect	70.26	77.78	72.87
OPERB	98.26	96.53	98.08	/	-	-	-	/	-	-	-

Table 6. The final compression ratios of algorithms in data aging that set $\epsilon_1 = 40m$ (or $\epsilon_1 = 30^\circ$ when using DAD) in the first run and $\epsilon_2 = 60m$ (or $\epsilon_2 = 50^\circ$ when using DAD) in the second run.

Alg. (PED)	UCar	Geolife	Mopsi	Alg. (SED)	UCar	Geolife	Mopsi	Alg. (DAD)	UCar	Geolife	Mopsi
DP	4.67	1.94	1.68	DP	10.03	3.83	2.78	DP	12.75	22.20	20.34
TP	4.66	2.07	1.74	TP	10.09	3.78	2.76	TP	3.80	13.28	11.30
BQS	4.13	1.67	1.43	SQUISH-E	11.38	4.55	3.47	OPW	4.09	13.88	11.92
SIPED (ϵ)	4.15	1.69	1.42	CISED (ϵ)	9.12	3.28	2.45	Interval	3.81	13.37	11.47
SIPED ($\frac{\epsilon}{2}$)	4.65	1.94	1.68	CISED ($\frac{\epsilon}{2}$)	10.66	3.94	2.99	Intersect	5.37	17.11	15.05
OPERB	5.03	2.03	1.97	/	-	-	-	/	-	-	-

Table 7. The compression ratios of algorithms running on the raw trajectories that set $\epsilon_3 = \epsilon_1 + \epsilon_2 = 100m$ (or $\epsilon_3 = 80^\circ$ when using DAD).

Alg. (PED)	UCar	Geolife	Mopsi	Alg. (SED)	UCar	Geolife	Mopsi	Alg. (DAD)	UCar	Geolife	Mopsi
DP	3.64	1.17	1.39	DP	7.44	1.91	2.69	DP	8.25	15.17	17.04
TP	3.70	1.22	1.50	TP	7.56	1.73	2.70	TP	2.47	7.51	9.42
BQS	3.23	1.71	1.39	SQUISH-E	10.36	2.82	4.10	OPW	3.23	8.66	10.75
SIPED (ϵ)	3.25	1.04	1.28	CISED (ϵ)	6.68	1.49	2.35	Interval	2.79	8.01	9.95
SIPED ($\frac{\epsilon}{2}$)	3.81	1.29	1.55	CISED ($\frac{\epsilon}{2}$)	8.02	1.87	2.90	Intersect	4.32	11.75	13.61
OPERB	4.08	1.43	1.56	/	-	-	-	/	-	-	-

final error bound, which is $\max\{\epsilon_1, \epsilon_2\}$, less than $\epsilon_3 = \epsilon_1 + \epsilon_2$; while for the other algorithms, they lose some compression ratios because of data aging.

Analyses of LS algorithms. DP is the only algorithm that is aging friendly *w.r.t.* PED and SED. Indeed, aging friendly is a result of the specific nature of algorithm DP, *i.e.*, batch and top-down, that always splits a trajectory into two sub trajectories by the same splitting point when it uses any PED or SED larger than the error bound.

Analyses of LS distance metrics. DAD is not aging friendly *w.r.t.* to any algorithm. It is different with PED and SED in that, the DAD of a point is closely related to its neighbor point while PED and SED are not, thus, once its neighbor point is removed, its DAD is also changed. This character makes it not aging friendly *w.r.t.* the DP algorithm.

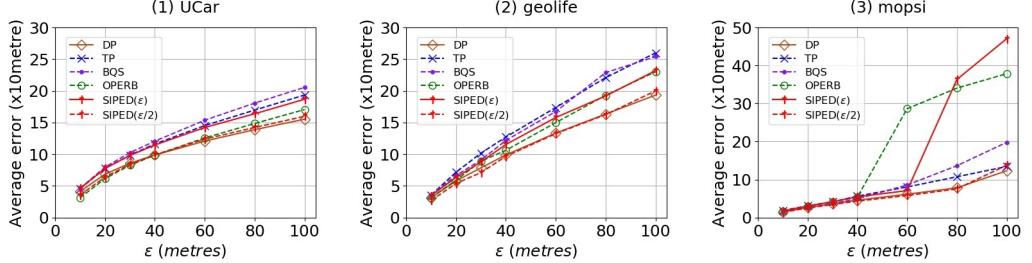
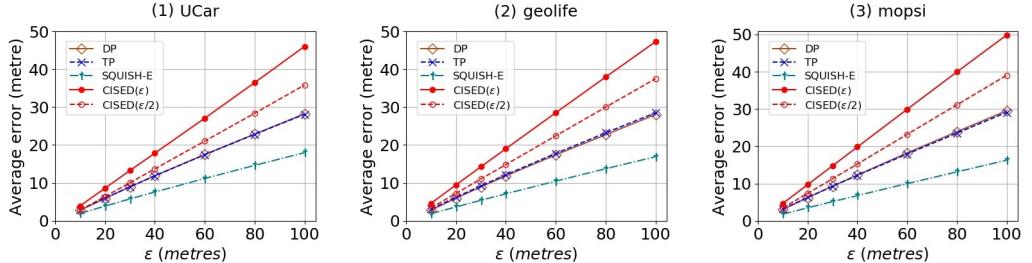
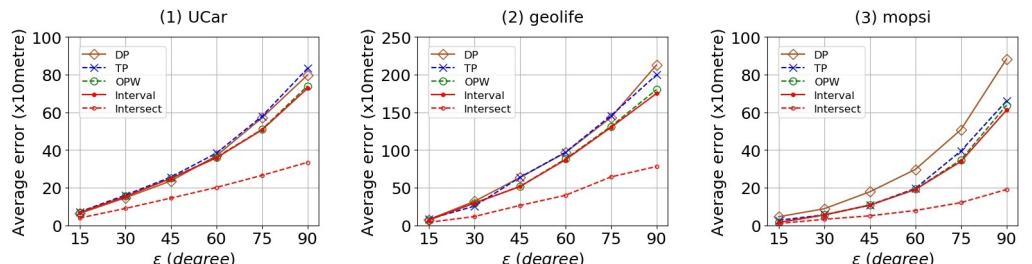
6.3.5 Evaluation and Analyses of Queries Friendliness. We finally evaluate those compressed trajectories from the viewpoint of trajectory application, *i.e.*, spatio-temporal query. The well-known spatio-temporal queries are *where_at*, *when_at*, *range*, *nearest_neighbor* and *spatial_join* [2]. Among them, *where_at* query, *i.e.*, “the position P of a moving object at time t ”, is the foundation of *range* and *nearest_neighbor* queries. Hence, we choose it to evaluate compressed trajectories simplified by LS algorithms using PED, SED and DAD. As mentioned in [2], the answer to *where_at* query is the expected position P' of the moving object at time t . Indeed, it is the *synchronized point* of P when the query is performed on simplified trajectories.

We first compress these trajectories using PED, SED and DAD, respectively. Then, for each point P in an original trajectory \tilde{T} , we perform a query on each of its compressed trajectories taking time $P.t$ as input, and calculate the distance between the actual position P and the expected position P'

Table 8. The max errors of spatio-temporal queries on compressed trajectories: fixed $\epsilon = 40m$.

Alg. (PED)	UCar	Geolife	Mopsi	Alg. (DAD)	UCar	Geolife	Mopsi
DP	3.48×10^6	1.83×10^6	5.03×10^6	DP	2.76×10^6	7.91×10^5	4.87×10^5
TP	3.51×10^6	1.91×10^6	5.03×10^6	TP	2.76×10^6	7.91×10^5	5.01×10^5
BQS	3.51×10^6	1.94×10^6	1.40×10^6	OPW	2.76×10^6	7.91×10^5	5.01×10^5
SIPED (ϵ)	3.51×10^6	1.02×10^6	1.39×10^6	Interval	2.76×10^6	7.91×10^5	5.01×10^5
SIPED ($\epsilon/2$)	3.50×10^6	1.02×10^6	1.39×10^6	Intersect	2.76×10^6	7.91×10^5	4.18×10^5
OPERB	3.50×10^6	1.56×10^6	1.39×10^6	/	-	-	-

Note that all algorithms using SED have the max query errors not more than the error bound, *i.e.*, $\epsilon = 40m$ here.

Fig. 33. Evaluation of spatio-temporal queries (PED) on full datasets: varying error bound ϵ .Fig. 34. Evaluation of spatio-temporal queries (SED) on full datasets: varying error bound ϵ .Fig. 35. Evaluation of spatio-temporal queries (DAD) on full datasets: varying error bound ϵ .

to denote the error of queries. The max and average errors of the queries are reported in Table 8 and Figures 33, 34, 35, 36, 37 and 38, respectively.

- (1) When using PED, the max query errors of all algorithms are more than 10^6 meters in all datasets, significantly larger than the error bound (40 meters in Table 8). The large max errors also lead to larger average query errors, *i.e.*, they are greater than error bounds in all datasets.
- (2) When using SED, the max query errors of all algorithms are clear not more than the error bounds, and the average query errors are consistent with those compression errors shown in Section 6.3.2.

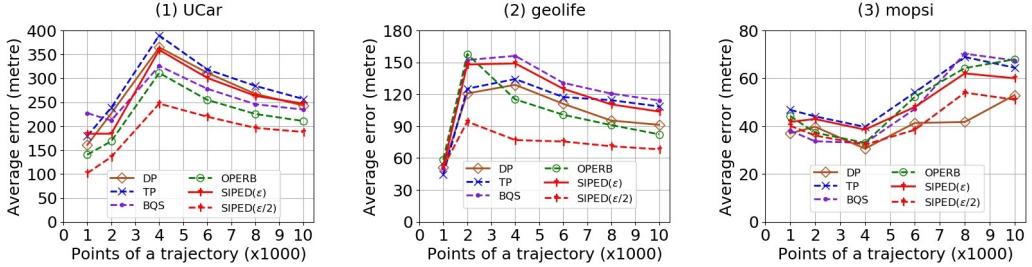


Fig. 36. Evaluation of spatio-temporal queries (PED) on small datasets: varying the size of trajectories.

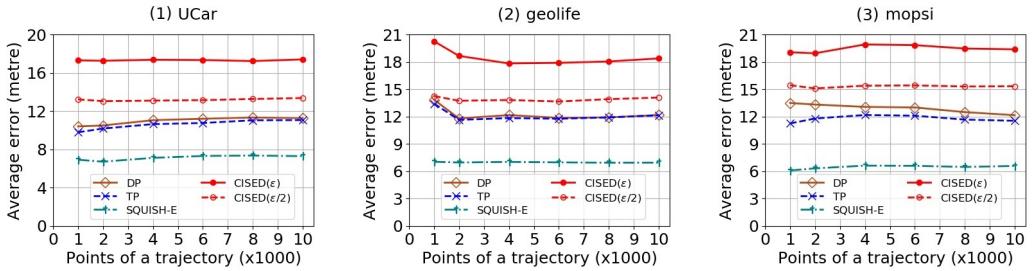


Fig. 37. Evaluation of spatio-temporal queries (SED) on small datasets: varying the size of trajectories.

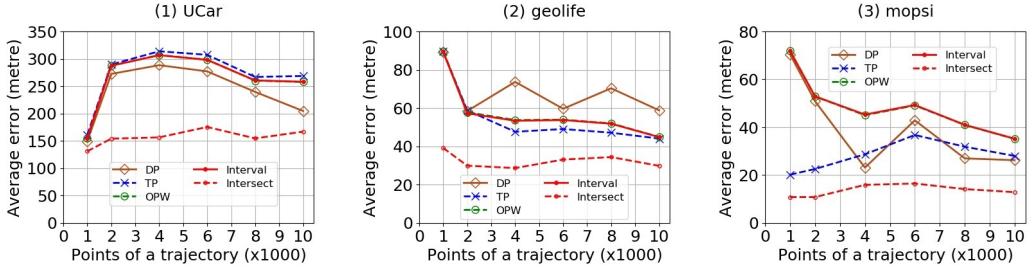


Fig. 38. Evaluation of spatio-temporal queries (DAD) on small datasets: varying the size of trajectories.

(3) When using DAD, the max query errors of all algorithms are more than 10^6 meters in dataset UCar, and more than 10^5 meters in datasets Geolife and Mopsi, respectively, also significantly larger than the error bound (40 meters in Table 8). Moreover, compared with using PED, it has more points having query errors larger than error bounds, thus, it has the largest average query errors in all algorithms and all datasets.

Analyses of LS algorithms. It is hard to tell regular conclusions of query errors of LS algorithms. Roughly speaking, algorithms Intersect using DAD and SQUISH-E using SED have obviously smaller query errors than other algorithms using the same distance metrics, partially because they have poorer compression ratios; and the average query errors of algorithms OPERB and SIPED (ϵ) using PED will remarkably increase when the error bounds are larger than certain value, e.g., 40 meters, in dataset Mopsi, because they may include points that lead to very large query errors.

Analyses of distance metrics. SED is spatio-temporal query friendly while the others are not. Given the same error bound, the average query errors of algorithms using PED are enormously larger than those using SED, brought by some points having extremely large query errors, and they are typically larger than error bounds. Similar to PED, algorithms using DAD also lead to enormously large query errors. Indeed, the temporal information is lost when using PED and DAD,

thus, they are not suitable for compressing trajectories in the context that spatio-temporal queries will run on the simplified trajectories.

6.3.6 Summary. From these tests we find the followings.

LS Algorithms. (1) The optimal algorithms have the best compression ratios, large average errors and the worst efficiencies. (2) Batch algorithms, except DP using DAD, have good compression ratios, normal average errors and poor efficiency. The bottom-up (TP) and top-down (DP) algorithms have the similar compression ratios and average errors when using either PED or SED. The bottom-up method has obviously better compression ratios than the top-down method when using DAD. The running time of batch algorithms DP and TP decreases and increases with the increase of error bound ϵ , respectively. When using PED or SED, top-down algorithm DP usually runs faster than bottom-up algorithm TP when the error bound ϵ is large (e.g., in Geolife, $\epsilon > 10$ metres when using PED and $\epsilon > 30$ metres when using SED). When using DAD, the top-down algorithm is normally a bit slower than the bottom-up algorithm. Top-down algorithms also run faster than bottom-up algorithms in high sampling datasets when using PED or SED. (3) Online algorithms OPW and BQS usually have better compression ratios than batch algorithms, the worst average errors, and poorer efficiency than batch algorithms. Algorithm SQUISH-E is on the other side of OPW and BQS. (4) One-pass algorithms OPERB, SIPED, CISED, Intersect and Interval have good compression ratios (comparable with the best sub-optimal algorithms), poor average errors and the best efficiency. The full ϵ sector/cone/range combining with a position/direction constraint always have better compression ratios and also larger average errors than the half ϵ sector/cone/range. One-pass algorithms show a linear running time and they are not very sensitive to error bound ϵ , and also scale well with the increase of trajectory sizes. (5) Algorithms DP using PED and SED are aging friendly, while others are not. All the tested algorithms have bounded errors in data aging, where the error bounds of DP using PED and SED are $\max\{\epsilon_1, \epsilon_2\}$ and the others are $\epsilon_1 + \epsilon_2$.

Distance Metrics. (1) The output sizes of algorithms using SED are approximately twice of PED, and in practice (e.g., $\epsilon < 100$ meters and $\epsilon < 60$ degrees), PED and SED usually bring obvious better compression ratios than DAD, especially in high sampling data sets. (2) The average errors of algorithms using SED are a bit larger than using PED. (3) Simplification using DAD is in general faster than PED and SED, and, indeed, the computation time of PED and SED is 2.3 and 1.7 times of DAD, respectively. (4) DAD is not aging friendly w.r.t. any tested algorithm. (5) SED is spatio-temporal queries friendly while the others are not. Note that algorithms using PED and/or DAD may lead to very large query errors.

7 CONCLUSIONS

Using three real-life trajectory datasets, we have systematically evaluated and analyzed error bounded LS algorithms for trajectory compression, including *both compression optimal and sub-optimal methods that use PED, SED and/or DAD*, in terms of compression ratios, errors, efficiency, aging friendliness and query friendliness.

Our experimental studies and analyses show the following.

(1) Choice of LS algorithms. Optimal algorithms bring the best compression ratios, however, their efficiency is obviously poorer than sub-optimal algorithms. The optimal simplified trajectory algorithms are essentially impractical from the perspective of applications, especially in cases when the input data set is large or computing resources are limited.

For compression sub-optimal algorithms, the output sizes of algorithms BQS and SIPED (ϵ) using PED, CISED (ϵ) using SED, and TP and Interval using DAD are approximately 103%–124%, 102%–115% and 102%–107% of the optimal algorithms, respectively. Batch algorithms using PED and SED

also have good compression ratios. Thus, they are the alternates of the optimal algorithms. More specifically, in case compression ratios are the first consideration, then algorithms BQS and SIPED (ϵ) using PED, CISED (ϵ) using SED, and TP and Interval using DAD are good candidates. In case average errors are concerned, then batch algorithms are good candidates as one-pass algorithms and online algorithms OPW and BQS all have relatively large average errors. In case running time is the most important factor or computing resources are limited, then one-pass algorithms are the best candidates. Indeed, one-pass algorithms run fast and require fewer resources, and have comparable compression ratios compared with batch and online algorithms. Hence, they are prominent trajectory compression algorithms when average errors are not the main concern.

Algorithms DP using PED and SED are aging friendly, which makes them have a bit better compression ratios in data aging given the same final error bound. Hence, DP is a good choice in scenario of data aging when compression ratios is the first consideration. Also remember that in data aging, each run of algorithm DP should take as input the whole raw/simplified trajectory and these trajectories must have the same start and end data points, otherwise, the *aging friendliness* of them would not be guaranteed.

(2) Choice of distance metrics. Users essentially choose a distance metric of PED, SED and DAD based on the needs of applications, e.g., SED is the only distance metric that is query friendly, hence, it is the best choice for applications like spatio-temporal queries. Further, the choice of a distance metric has impacts on the performance. For compression ratios, the using of synchronized distance SED saves temporal information of trajectories with a cost of approximately double-sized outputs compared with using PED in all datasets; PED has obvious better compression ratios than DAD in all datasets, and SED is also better than DAD in high sampling datasets. For efficiency, the computation time of PED and SED is 2.3 and 1.7 times of DAD, respectively.

APPENDIX A: ADDITIONAL LINE SIMPLIFICATION ALGORITHMS

Apart from the techniques evaluated in this article, there exist other approaches for various requirements of trajectory compression. We next summarized some of the representative LS algorithms. Interested readers may refer to [14, 25, 33, 41] for more details.

Weak simplification algorithms. Weak simplification allows that the output data points may not belong to the original data sets [37] (otherwise, the strong simplification). That is, weak simplification allows data interpolation. Algorithms Sleeve [42], OPERB [15] and CISED [14] are both strong and weak LS algorithms whose weak versions have better compression ratios than their strong counterparts, and even comparable with the optimal algorithms [14].

Algorithms using other error metrics. A number of algorithms [1, 3, 5, 40] have been developed to solve the “min-#” problem under alternative error metrics. [1] studies the problem under the L_1 and uniform (also known as Chebyshev) metric. [5] defines the integral square synchronous Euclidean distance (ISSED), an accumulation of square SED, and uses it to speed up the construction of reachability graphs, [40] follows the ideas of [5] and Dots [3] is an online trajectory simplification algorithm that uses ISSED. Note that these error metrics are not as popular as PED and SED.

Algorithms on the min- ϵ problem. Some studies focus on the min- ϵ problem [4] that, given m , constructs an approximate curve consisting of at most m line segments with the minimum error. SQUISH(λ) [24] and SQUISH-E(λ) [25] are such algorithms that compress a trajectory of length n into a trajectory of length at most n/λ . These methods lack the capability of compressing trajectories while ensuring that SED errors are within a user-specified bound [25].

REFERENCES

- [1] AGARWAL, P. K., AND VARADARAJAN, K. R. Efficient algorithms for approximating polygonal chains. *Discrete and Computational Geometry* 23 (2000), 273–291.
- [2] CAO, H., WOLFSON, O., AND TRAJCEVSKI, G. Spatio-temporal data reduction with deterministic error bounds. *VLDBJ* 15, 3 (2006), 211–228.
- [3] CAO, W., AND LI, Y. Dots: An online and near-optimal trajectory simplification algorithm. *Journal of Systems and Software* 126(Supplement C) (2017), 34–44.
- [4] CHAN, W., AND CHIN, F. Approximation of polygonal curves with minimum number of line segments or minimal error. *IJCGA* 6, 1 (1996), 378–387.
- [5] CHEN, M., XU, M., AND FRÄNTI, P. A fast multiresolution polygonal approximation algorithm for GPS trajectory simplification. *TIP* 21, 5 (2012), 2770–2785.
- [6] CHEN, Y., JIANG, K., ZHENG, Y., LI, C., AND YU, N. Trajectory simplification method for location-based social networking services. In *LBSN* (2009).
- [7] DOUGLAS, D. H., AND PEUCKER, T. K. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer* 10, 2 (1973), 112–122.
- [8] DUNHAM, J. G. Piecewise linear approximation of planar curves. *PAMI* 8 (1986).
- [9] HERSHBERGER, J., AND SNOEYINK, J. Speeding up the douglas-peucker line-simplification algorithm. *Technical Report, University of British Columbia* (1992).
- [10] IMAI, H., AND IRI, M. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics, and Image Processing* 36 (1986), 31–41.
- [11] KE, B., SHAO, J., AND ZHANG, D. An efficient online approach for direction-preserving trajectory simplification with interval bounds. In *MDM* (2017).
- [12] KEOGH, E. J., CHU, S., HART, D. M., AND PAZZANI, M. J. An online algorithm for segmenting time series. In *ICDM* (2001).
- [13] LANGE, R., DÜRR, F., AND ROTHERMEL, K. Efficient real-time trajectory tracking. *VLDBJ* 20, 5 (2011), 671–694.
- [14] LIN, X., JIANG, J., MA, S., ZUO, Y., AND HU, C. One-pass trajectory simplification using the synchronous euclidean distance. *VLDBJ* 28, 6 (2019), 897–921.
- [15] LIN, X., MA, S., ZHANG, H., WO, T., AND HUAI, J. One-pass error bounded trajectory simplification. *PVLDB* 10, 7 (2017), 841–852.
- [16] LIU, J., ZHAO, K., SOMMER, P., SHANG, S., KUSY, B., AND JURDAK, R. Bounded quadrant system: Error-bounded trajectory compression on the go. In *ICDE* (2015).
- [17] LONG, C., WONG, R. C.-W., AND JAGADISH, H. Direction-preserving trajectory simplification. *PVLDB* 6, 10 (2013), 949–960.
- [18] LONG, C., WONG, R. C.-W., AND JAGADISH, H. Trajectory simplification: On minimizing the direction-based error. *PVLDB* 8, 1 (2014), 49–60.
- [19] MARIESCU-ISTODOR, R., AND FR?NTI, P. Grid-based method for gps route analysis for retrieval. *ACM Transactions on Spatial Algorithms and Systems* 3, 3 (2017).
- [20] MELKMAN, A., AND O'ROURKE, J. On polygonal chain approximation. *Machine Intelligence and Pattern Recognition* 6 (1988), 87–95.
- [21] MERATNIA, N., AND DE BY, R. A. Spatiotemporal compression techniques for moving point objects. In *EDBT* (2004).
- [22] METHA, R., AND V.K.MEHTA. *The Principles of Physics*. S Chand, 1999.
- [23] MUCKELL, J., HWANG, J., LAWSON, C. T., AND RAVI, S. S. Algorithms for compressing GPS trajectory data: an empirical evaluation. In *ACM-GIS* (2010).
- [24] MUCKELL, J., HWANG, J., PATIL, V., LAWSON, C. T., PING, F., AND RAVI, S. S. SQUISH: an online approach for GPS trajectory compression. In *COM.Geo* (2011).
- [25] MUCKELL, J., OLSEN, P. W., HWANG, J.-H., LAWSON, C. T., AND RAVI, S. S. Compression of trajectory data: a comprehensive evaluation and new approach. *GeoInformatica* 18, 3 (2014), 435–460.
- [26] NIBALI, A., AND HE, Z. Trajic: An effective compression system for trajectory data. *TKDE* 27, 11 (2015), 3138–3151.
- [27] PAVLIDIS, T., AND HOROWITZ, S. L. Segmentation of plane curves. *TOC* 23, 8 (1974), 860–870.
- [28] POPA, I. S., ZEITOUNI, K., VINCENTORIA, AND KHARRAT, A. Spatio-temporal compression of trajectories in road networks. *GeoInformatica* 19, 1 (2014), 117–145.
- [29] RAMER, U. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing* 1 (1972), 244–256.
- [30] REUMANN, K., AND WITKAM, A. Optimizing curve segmentation in computer graphics. In *International Computing Symposium* (1974).
- [31] RICHTER, K.-F., SCHMID, F., AND LAUBE, P. Semantic trajectory compression: Representing urban movement in a nutshell. *JOSIS* 4, 1 (2012), 3–30.
- [32] SCHMID, F., RICHTER, K., AND LAUBE, P. Semantic trajectory compression. In *SSTD* (2009).

- [33] SHI, W., AND CHEUNG, C. Performance evaluation of line simplification algorithms for vector generalization. *Cartographic Journal* 43, 1 (2006), 27–44.
- [34] SKLANSKY, J., AND GONZALEZ, V. Fast polygonal approximation of digitized curves. *Pattern Recognition* 12 (1980), 327–331.
- [35] SONG, R., SUN, W., ZHENG, B., AND ZHENG, Y. Press: A novel framework of trajectory compression in road networks. *PVLDB* 7, 9 (2014), 661–672.
- [36] TOUSSAINT, G. T. On the complexity of approximating polygonal curves in the plane. In *International Symposium on Robotics and Automation* (1985).
- [37] TRAJCEVSKI, G., CAO, H., SCHEUERMANN, P., WOLFSONZ, O., AND VACCARO, D. On-line data reduction and the quality of history in moving objects databases. In *MobiDE* (2006).
- [38] VAN HUNNIK, R. Extensive comparison of trajectory simplification algorithms. master thesis, 2017.
- [39] WILLIAMS, C. M. An efficient algorithm for the piecewise linear approximation of planar curves. *Computer Graphics and Image Processing* 8 (1978), 286–293.
- [40] WU, F., FU, K., WANG, Y., AND XIAO, Z. A graph-based minimal number and error-optimal trajectory simplification algorithm and its extension towards online services. *ISPRS International Journal of Geo-Information* 6, 19 (2017).
- [41] ZHANG, D., DING, M., YANG, D., LIU, Y., FAN, J., AND SHEN, H. T. Trajectory simplification: An experimental study and quality analysis. *PVLDB* 9, 11 (2018), 934–946.
- [42] ZHAO, Z., AND SAALFELD, A. Linear-time sleeve-fitting polyline simplification algorithms. In *AutoCarto* (1997).
- [43] ZHENG, Y., XIE, X., AND MA, W. GeoLife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.* 33, 2 (2010), 32–39.