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

XUELIAN LIN, SHUAI MA*, JIAHAO JIANG, YANCHEN HOU AND TIANYU WO, State Key Laboratory of Software Development Environment, Beihang University, China

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 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 errors. 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 provides a full picture of error bounded line simplification algorithms, which leads to guidelines on how to choose appropriate algorithms and distance metrics for practical applications.

CCS Concepts: • **Information systems → Spatial-temporal systems**; • **Theory of computation → Data compression**.

Additional Key Words and Phrases: trajectory compression, line simplification, batch algorithms, online algorithms, one-pass algorithms

ACM Reference Format:
Xuelian Lin, Shuai Ma*, Jiahao Jiang, Yanchen Hou and Tianyu Wo. 2021. Error Bounded Line Simplification Algorithms for Trajectory Compression: An Experimental Evaluation. 1, 1 (July 2021), 43 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 [3, 8, 9, 25, 31, 33, 39, 43, 44, 47, 52, 53], and trajectory compression techniques [3, 4, 7–9, 12, 16, 17, 20, 23, 25, 27, 29–33, 39, 42–44, 48, 52, 54, 56, 59] have been developed to alleviate this situation.

Author's address: Xuelian Lin, Shuai Ma*, Jiahao Jiang, Yanchen Hou and Tianyu Wo, State Key Laboratory of Software Development Environment, Beihang University, 37 Xueyuan Rd, Haidian Dist., Beijing, China, 100191, emails:{linxl,mashuai, jianjh,houyc,woty}@buaa.edu.cn.

XXXX-XXXX/2021/7-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Due to the limitations (poor compression ratios and data reconstruction overheads) of lossless compression, lossy compression techniques have become the mainstream of trajectory compression [30, 66]. Quite a few lossy trajectory compression techniques, most notably the piece-wise line simplification [3, 4, 7–9, 16, 23, 25, 27, 29–33, 39, 42, 43, 54, 59], have been developed. The idea of piece-wise line simplification (LS) comes from graphic and computational geometry, whose target is to approximate a fine piece-wise linear curve with a coarse one. A planar curve is basically defined with a sequence of data points in computer graphics [64]. If all data points of the coarse curve are a subset of the original curve, then it is referred to as *strong simplification*; otherwise, it is referred to as *weak simplification* [59]. LS has two optimization problems [5, 22, 46], (1) the *min-#* problem: given a curve and an error bound ϵ, to construct an approximate curve with errors bounded in ϵ and having the minimum number of line segments [5, 22]. Most LS algorithms for trajectory compression [3, 4, 7–9, 12, 16, 20, 23, 25, 27, 29–33, 39, 43, 59] are for this problem, and (2) the *min-ϵ* problem: given a curve and a positive integer m, construct an approximate curve consisting of at most m line segments with the minimum error [5, 22]. Trajectory compression algorithms like SQUISH (λ) [42] and SQUISH-E (λ) [43] solve this problem and are the complementary ones to those solving the *min-#* problem. Line simplification has a wide usage due to its beneficial features: (a) simple and easy to implement, (b) light and applicable to resource-constrained devices, and (c) bounded errors (algorithms for the *min-#* problem) with good compression ratios. It is a great advantage that LS is suitable to run on resource-constrained end devices, such that the trajectories could be simplified at the early time, to save not only the storage of data servers, but also the network bandwidth between the end devices and the data servers. LS could also be combined with other techniques, such as dilution-matching-encoding [17] that maps the simplified trajectories to road networks, and discovers the high frequency patterns of compressed trajectories to further improve the effectiveness of trajectory compression. In this article, we focus on the error bounded LS algorithms that solve the *min-#* problem and require no extra knowledge, together with a brief discussion of semantic based compression methods. Please also refer to [42, 43, 66] for the algorithms that solve the *min-ϵ* problem, which are essentially not error bounded.

In this article, we interchangeably use the terms "compression" and "simplification" without ambiguity as much existing literature [29, 31, 39, 42, 43, 66]. Indeed, simplification is quite older than computational geometry and data compression because it traces its origins in the cartography [61, 62], a few centuries before inventing the computers. In its original sense, simplification is but one aspect of the problem called "data generalization" [58, 61, 62] in cartography, where in practice one would deliberately change the outcomes of a compression/simplification because certain properties need to be preserved with appropriate visibility.

Algorithm taxonomy. LS algorithms fall into two categories: *compression optimal* and *compression sub-optimal* algorithms. *Compression optimal* methods [5, 22] 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. Following the taxonomy shown in [27, 29, 30], compression sub-optimal algorithms can further be classified into batch, online and one-pass algorithms, which correspond to offline, online and real-time algorithms, respectively, in [27].

(1) *Batch algorithms* are divided into top-down methods, *e.g.,*Ramer [50] and Douglas-Peucker (DP) [12, 39], and bottom-up methods, *e.g.,* Theo-Pavlidis (TP) [46], 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 [39], SQUISH-E [43] and BQS [31], 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 [30], SIPED [13, 55, 63, 68], CISED [29], Intersect [33] and Interval [23], 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, and ensure that the maximal distance from the output trajectory to the input trajectory is bounded by a PED error bound. Here a trajectory is basically treated as a sequence of spatial data points. LS *algorithms using* PED *bring good compression ratios [3, 9, 12, 20, 31, 43, 54] at a cost of losing temporal information of trajectories*. Hence, PED is *not spatio-temporal query friendly*, *i.e.,* a spatio-temporal query like *where_at* [3] on such a simplified trajectory may return a point with a distance great larger than the PED error bound adopted in the simplification algorithm. However, PED remains useful, *e.g.,* trajectory simplification using PED can serve as a pre-processing step of trajectory clustering that is the base of applications like traffic pattern recognition and urban planning [37, 67], where the shapes of trajectories rather than the detailed positions of individuals are concerned.

SED is then introduced to preserve the temporal information [3, 39]. 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* [3]. The algorithms ensure that the *maximal* SED from the output trajectory to the input trajectory is bounded by an SED error bound. SED friendly supports applications such as spatio-temporal queries, *i.e.,* a spatio-temporal query like *where_at* [3] on such a simplified trajectory returns the expected (synchronized) point that has a (SED) distance less than the error bound used in the simplification algorithms. Obviously, given the same error bound, algorithms using SED typically produce more points than PED as they further preserve the temporal information.

DAD is introduced to preserve the direction information of moving objects [33, 66], and is initially called the *direction-based measurement* in [33]. It is important for applications such as trajectory clustering and direction-based query processing [33, 34]. Different from PED and SED, DAD is the direction deviation of a moving object, measured by angles rather than Euclidean distances. However, the temporal information is not preserved for DAD. Hence, it is not friendly for spatio-temporal queries as well.

Quality criteria. Quality criteria are needed to evaluate LS algorithms from two levels. The first level comes from LS itself, including compression ratios, efficiency and simplification errors: (1) *compression ratios* are the ratios of the data sizes of the simplified trajectories to the original trajectories, (2) *efficiency* is the time taken by LS algorithms to compress trajectories, and (3) *simplification errors* are the *maximal* and *average* distances between the simplified trajectories and the original trajectories. They are commonly used for quality evaluations, *e.g.,* [29, 31, 33, 43, 66].

The second level comes from applications, *e.g.,* a "where_at" query [3, 60] concerns the errors of its answer, and a trajectory clustering method concerns the similarity between the clusters of

Table 1. Error bounded trajectory simplification algorithms

Category		Algorithms	PED	SED	DAD	Time	Space	Rep	Key Ideas
optimal		Optimal [22]	✓	✓	✓	$O(n^3)$	$O(n^2)$	✓	reachability graph
		OptPED [5]	✓	×	×	$O(n^2)$	$O(n^2)$		reachability graph & sector intersection
		OptLISSED [8]	×	✓*	×	$O(n^2)$	$O(n^2)$		reachability graph & LISSED
		SP [33]	×	×	✓	$O(n^2)$	$O(n^2)$		reachability graph & range intersection
sub-optimal	batch	Ramer [50]	✓	✓	✓	$O(n^2)$	$O(n)$		top-down
		DP [12, 39]	✓	✓	✓	$O(n^2)$	$O(n)$	✓	top-down
		TP [46]	✓	✓	✓	$O(n^2/K)$	$O(n)$	✓	bottom-up
		MRPA [8]	×	✓*	×	$O(n)$	$O(n)$		LISSED & bottom-up
	online	OPW [39]	✓	✓	✓	$O(n^2)$	$O(n)$	✓	top-down in opening window
		BQS [31]	✓	×	×	$O(n^2)$	$O(\|n\|)$	✓	top-down, window & convex hull
		SWAB [25]	✓	✓	✓	$O(n * \|Q\|)$	$O(\|Q\|)$		bottom-up in sliding window
		SQUISH-E [43]	×	✓	×	$O(n \log \|Q\|)$	$O(\|Q\|)$	✓	bottom-up & priority queue
		DOTS [4]	×	✓*	×	$O(n^2/K)$	$O(\|Q\|^2)$	✓	LISSED & incremental DAG
	one-pass	RW [51]	✓	×	×	$O(n)$	$O(1)$		strip
		LDR [27, 59]	×	✓	×	$O(n)$	$O(1)$		linear dead reckoning
		OPERB [30]	✓	×	×	$O(n)$	$O(1)$	✓	fitting function
		SIPED [13, 68]	✓	×	×	$O(n)$	$O(1)$	✓	sector intersection
		CISED [29]	×	✓	×	$O(n)$	$O(1)$	✓	spatio-temporal cone intersection
		Intersect [33]	×	×	✓	$O(n)$	$O(1)$	✓	range intersection with $\frac{\epsilon}{2}$-range
		Interval [23]	×	×	✓	$O(n)$	$O(1)$	✓	range intersection with ϵ-range

Here (1) K is the number of the final segments of a trajectory, $|Q|$ is the size of a buffer/window and "Rep" means "representative" (we comprehensively consider the performance of algorithms, the supporting of distance metrics and the novelty of their key ideas to choose the representatives), (2) algorithms OptLISSED, MRPA and DOTS alternatively use Local Integral Square SED (LISSED [8]) as the error measure instead of directly using SED, and among them, DOTS is the representative, (3) algorithms Ramer[50] and Douglas-Peucker (DP) [12, 39] were independently developed with an extremely similar idea. Thus, they are also jointly referred to as "Ramer-Douglas-Peucker", (4) one-pass algorithms OPERB and CISED both have strong and weak versions [29, 30], where the former only allow the data points belonging to the original trajectory and the later allows data interpolations in the simplified trajectories, and (5) batch and top-down algorithms DP [12, 39] and Ramer [50] are aging friendly, while other algorithms are not.

the original and simplified trajectories. Indeed, each application may have its own point of view for evaluations. Here we choose *spatio-temporal queries* [3, 60] as the representatives of trajectory applications because they are commonly used in the management of trajectories. We use *query* errors to evaluate the qualities, which are the temporal differences (*e.g.,* "when_at") or spatial distances (*e.g.,* "where_at") of the answers between the original and simplified trajectories.

Motivations. Empirical studies of trajectory compression algorithms have been conducted [41, 43]. However, they only discuss a small number of algorithms. The very recent study [66] 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, aging friendliness) are not systematically studied. That is, the impacts of both error bounds and the sizes of trajectories on running time and the impacts of error bounds on compression ratios are not fairly evaluated, and the *data aging* problem is not investigated at all. The simplified trajectories are stored in data stores, as time goes by, less precision may become acceptable for old trajectories [3], and they may 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 [3].

Moreover, certain important algorithms are not evaluated in [66], *e.g.,* optimal algorithms using PED and SED, and one-pass algorithms SIPED and CISED. Indeed, algorithm SIPED is *completely overlooked* by existing trajectory compression studies as it is originally developed in fields of computational geometry and pattern recognition [13, 55, 63, 68], and we recently find it can be

easily adopted for trajectory compression with good performance. That is to say, a systematic experimental evaluation of line simplification algorithms for trajectory compression remains necessary for choosing the appropriate algorithms and distance metrics for practical applications.

Contributions. In this article, we conduct a systematic experimental evaluation and analysis 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 view of *aging friendliness*, and prove that (a) 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, where ϵ_1 and ϵ_2 are the error bounds set in the first and second times of simplification, respectively, and (b) 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 analysis of error bounded trajectory simplification algorithms. In comparison with the recent study [66], (1) for a fair running time analysis, all algorithms are (re)-implemented in Java, unlike [66] that reports running time of algorithms with different programming languages, (2) compression ratio analyses are systematically considered, (3) variations of distance metrics are studied, (4) optimal algorithms using PED and SED and one-pass algorithms SIPED and CISED are investigated, (5) weak simplification algorithms OPERB-A and CISED-W are studied, and (6) data aging of LS algorithms is studied. Some new findings are summarized in Section 6.3.6.

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 trajectory compression methods 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.

Point. A data point is defined as a triple $P(x, y, t)$ in a suitable coordinate system, representing that a moving object is located at (x, y) at time t. In practice, a raw data point collected from an end device of some Global Navigation Satellite System is usually defined by geo-spatial data standards (e.g., World Geodetic System 84 coordinates) with a format of (*longitude, latitude, altitude, time*). In this article, these raw data points are projected to a x-y-t 3D Euclidean space (called map projection [19]) during trajectory simplification.

Table 2. Summary of notations

Notations	Semantics
P	a data point
$\ddot{\mathcal{T}}$	a trajectory $\ddot{\mathcal{T}}$ is a sequence of data points
$\overline{\mathcal{T}}$	a piece-wise line representation of a trajectory $\ddot{\mathcal{T}}$
$\mathcal{L}$	a directed line segment
$\mathcal{M}$	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

Trajectory. A *trajectory* $\ddot{\mathcal{T}}\,[P_0, \ldots, P_n]$ 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 $0 \le i < j \le n$). Intuitively, a trajectory is the path (or track) that a moving object follows through space as a function of time [40].

Directed line segment. A directed line segment (or line segment for simplicity) $\mathcal{L}$ is defined as $\overrightarrow{P_sP_e}$, 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 $\mathcal{L} = \overrightarrow{P_sP_e}$ 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* $\overline{\mathcal{T}}\,[\mathcal{L}_0, \ldots, \mathcal{L}_m]$ $(0 < m \le n)$ of a trajectory $\ddot{\mathcal{T}}\,[P_0, \ldots, P_n]$ 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 a simplified trajectory, essentially the sequence of end points of all line segments, 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* [39] (SED) and the *direction-aware distance*[33, 66] (DAD). Consider a data point P and a directed line segment $\mathcal{L} = \overrightarrow{P_sP_e}$.

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 $\overrightarrow{P_sP_e}$. This definition is also called the *tolerance-zone* error measure [1, 6, 11, 22, 38]. Note that in the field of computational geometry, (1) there is a slight variation of *tolerance-zones*, called the *infinite beam* or *parallel-strip* criterion [6, 11], which is the perpendicular Euclidean distance of a point to a line, and (2) it is believed that tolerance-zones could produce a compressed version that better captures the features of the original path/curve [1, 6, 11]. If not otherwise specified, we use tolerance-zones to evaluate algorithms.

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 $c = \frac{P.t - P_s.t}{P_e.t - P_s.t}$.

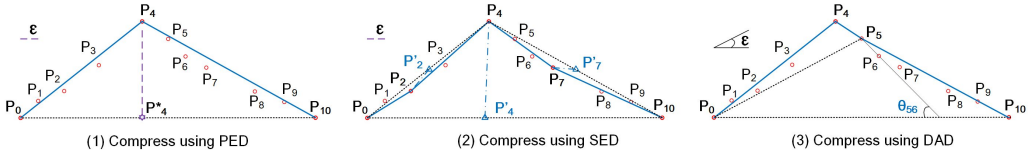

Fig. 1. A trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ with 11 points is compressed by the Douglas–Peucker algorithm [12] using distance metrics PED, SED and DAD, respectively.

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 $\frac{|\overrightarrow{P_sP_e}|}{P_e.t-P_s.t}$ between points P_s and P_e [3, 29]. Then the *synchronized point* P' of a point P *w.r.t.* the line segment $\overrightarrow{P_sP_e}$ is the expected position of the moving object on $\overrightarrow{P_sP_e}$ at time $P.t$, obtained by a linear interpolation [3]. More specifically, a synchronized point P'_i of P_i ($s \le i < e$) *w.r.t.* the line segment $\overrightarrow{P_sP_e}$ is a point on $\overrightarrow{P_sP_e}$ satisfying $|\overrightarrow{P_sP'_i}| = \frac{P_i.t-P_s.t}{P_e.t-P_s.t} \cdot |\overrightarrow{P_sP_e}|$, which means that the object moves from P_s to P_e at an average speed $\frac{|\overrightarrow{P_sP_e}|}{P_e.t-P_s.t}$, and its position at time $P_i.t$ is the point P'_i on $\overrightarrow{P_sP_e}$ having a distance of $\frac{P_i.t-P_s.t}{P_e.t-P_s.t} \cdot |\overrightarrow{P_sP_e}|$ to P_s [3, 8, 29, 39, 66].

Instead of directly using SED, [8] introduces Local Integral Square SED (LISSED, also called LSSD in [4, 8]) with $lissed(P_s, P_{s+k}) = \sum_{i=s}^{s+k-1} sed^2(P_i, \overrightarrow{P_sP_{s+k}})$. The most important feature of LISSED is that it can be computed in an incremental way, *i.e.,* given information about $lissed(P_s, P_{s+k})$, $lissed(P_s, P_{s+k+1})$ can be calculated in $O(1)$ time, such that algorithms using it (*e.g.,* MRPA [8] and DOTS [4]) have lower time complexities compared with directly using SED. It is worth pointing out that (1) LISSED is a special SED-based *error* measure rather than a new kind of *distance* metric, (2) if the LISSED error bound of such an algorithm is set to ϵ^2, then the maximal SED error between the original and simplified trajectories is always not greater than ϵ, and (3) the most recent algorithm directly using SED, *i.e.,* CISED, is very efficient, and has $O(n)$ time.

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 differences, 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) $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ is a trajectory having 11 data points, (2) the set of two continuous line segments $\{\overrightarrow{P_0P_4}, \overrightarrow{P_4P_{10}}\}$, the set of four continuous line segments $\{\overrightarrow{P_0P_2}, \overrightarrow{P_2P_4}, \overrightarrow{P_4P_7}, \overrightarrow{P_7P_{10}}\}$ and the set of three continuous line segments $\{\overrightarrow{P_0P_4}, \overrightarrow{P_4P_5}, \overrightarrow{P_5P_{10}}\}$ are three piece-wise line representations of trajectory $\ddot{\mathcal{T}}$, (3) $ped\left(P_4, \overrightarrow{P_0P_{10}}\right) = |\overrightarrow{P_4P^*_4}|$, where P^*_4 is the perpendicular point of P_4 *w.r.t.* line segment $\overrightarrow{P_0P_{10}}$, (4) For P_4, its synchronized point P'_4 *w.r.t.* $\overrightarrow{P_0P_{10}}$ satisfies $\frac{|\overrightarrow{P_0P'_4}|}{|\overrightarrow{P_0P_{10}}|} = \frac{P_4.t-P_0.t}{P_{10}.t-P_0.t} = \frac{4-0}{10-0} = \frac{2}{5}$, (5) $sed\left(P_4, \overrightarrow{P_0P_{10}}\right) = |\overrightarrow{P_4P'_4}|$, $sed\left(P_2, \overrightarrow{P_0P_4}\right) = |\overrightarrow{P_2P'_2}|$ and $sed\left(P_7, \overrightarrow{P_4P_{10}}\right) = |\overrightarrow{P_7P'_7}|$, 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 $\overrightarrow{P_0P_{10}}$, $\overrightarrow{P_0P_4}$ and $\overrightarrow{P_4P_{10}}$, respectively, and (6) $dad\left(\overrightarrow{P_5P_6}, \overrightarrow{P_0P_{10}}\right) = \theta_{56}$ is the DAD of line segments $\overrightarrow{P_5P_6}$ to $\overrightarrow{P_0P_{10}}$. □

Trajectory simplification algorithms. Given a trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ and a pre-specified bound ϵ, a *trajectory simplification algorithm* $\mathcal{A}$ using PED (respectively, SED and DAD) produces

a piece-wise line representation $\overline{\mathcal{T}}[\mathcal{L}_0, \ldots, \mathcal{L}_m]$ $(0 < m \leq n)$ by applying distance checking of PED (respectively, SED and DAD) with respect to ϵ, 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 $\ddot{\mathcal{T}}_i \left[P_{s_i}, \ldots, P_{e_i}\right]$ of $\ddot{\mathcal{T}}$.

Error bounded algorithms. Given a trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ and a pre-specified bound ϵ, trajectory simplification 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 $\ddot{\mathcal{T}}$, 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 $dad\left(\overrightarrow{P_k P_{k+1}}, \mathcal{L}_i\right)$) is no more than ϵ.

Note that here there is no need to 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 $\ddot{\mathcal{T}}$. That is to say, data interpolations are allowed.

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) [22] first formulates the *min-#* problem 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* [57] and *sector intersection* [38], faster optimal algorithms are proposed with an improved time complexity to $O(n^2 \log n)$. Further, [5] proves that the *min-#* problem (using PED) for a general polygonal curve can be solved in $O(n^2)$ time by using the *sector intersection* mechanism, and for some special curves (*e.g.,* a polygonal curve forming a part of a convex polygon), either open or closed (if there is an edge joining the first and the last points, then it is closed; otherwise, it is open), this problem can be solved in $O(n)$ time. Note that, in general, a trajectory is not necessary a convex or closed polygonal curve, instead, it is often open and concave. Thus, the best optimal algorithm for trajectory simplification using PED still has $O(n^2)$ time. OptLISSED [8] using LISSED also has a time complexity of $O(n^2)$, and algorithm SP [33] 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 directly, and Optimal remains the best optimal solution for SED. *As all the optimized algorithms have the same effectiveness when using the same distance metric, and essentially work for small size trajectories only, we choose algorithm* Optimal *that supports* PED*,* SED *and* DAD *as the representative of optimal* LS *algorithms.*

Given a trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ and an error bound ϵ, algorithm Optimal [22] solves the optimal trajectory simplification problem in two steps: (1) it first constructs a reachability graph G of $\ddot{\mathcal{T}}$, and then (2) searches a shortest path from point P_0 to point P_n in graph G. The reachability graph of a trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ *w.r.t.* an error bound ϵ is $G = (V, E)$, where (1) $V = \{P_0, \ldots, P_n\}$, 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 $\overrightarrow{P_s P_{s+k}}$ 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 $\ddot{\mathcal{T}}$. The path also reveals the subset of points of $\ddot{\mathcal{T}}$ 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 $\ddot{\mathcal{T}}$.

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 $\overrightarrow{P_s P_{s+k}}$ 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 $\overrightarrow{P_s P_{s+k}}$ 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,

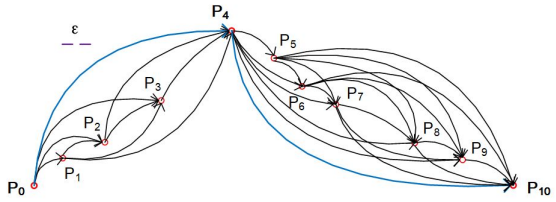

Fig. 2. Example of reachability graph of trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ whose shortest path is (P_0, P_4, P_{10}).

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.

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

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 [39].

(1) Top-down algorithms recursively divide a trajectory into sub-trajectories until the stopping condition is met. Algorithms Ramer [50] and Douglas-Peucker (DP) [12] support all the three distances PED, SED and DAD. Note that Ramer came about the same time as DP, but the authors of DP were not aware and independently developed an extremely similar idea which, although published (officially) later, became more widely popular, and the DP using SED is also called TD-TR [39]. An improved method of DP with a time complexity of $O(n \log n)$, based on *convex hulls*, is proposed in [20], 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. In each iteration, the distances of newly generated line segments are recalculated. To our knowledge, Theo-Pavlidis (TP) [46] is the only bottom-up batch LS algorithm that supports PED, SED and DAD. Besides, MRPA [8] using LISSED is also a bottom-up algorithm.

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 the famous top-down algorithm* DP *and the classic bottom-up algorithm* TP *that all support* PED, SED *and* DAD *as the representatives of batch* LS *algorithms.*

*4.1.1 Algorithm Douglas-Peucker (*DP*) [12].* 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.

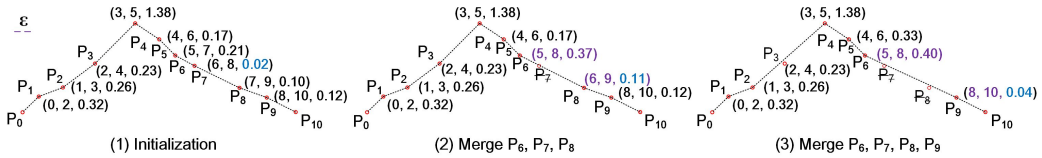

Fig. 3. The trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ is compressed by the Theo-Pavlidis algorithm using PED to two line segments. The triple $(i, j, cost)$ is the *cost* of merging the line segments $\overline{P_iP_t}$ and $\overline{P_tP_j}$.

Given a trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ and an error bound ϵ, algorithm DP uses the first point P_0 and the last point P_n of $\ddot{\mathcal{T}}$ 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 $ped(P_k, \mathcal{L}) = \max\{ped(P_0, \mathcal{L}), \ldots, ped(P_n, \mathcal{L})\} \leq \epsilon$, then it returns $\{\mathcal{L}(P_0, P_n)\}$. Otherwise, it divides $\ddot{\mathcal{T}}$ into two sub-trajectories $\ddot{\mathcal{T}}[P_0, \ldots, P_k]$ and $\ddot{\mathcal{T}}[P_k, \ldots, P_n]$, 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 $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ shown in Figure 1. The DP Algorithm firstly creates $\overrightarrow{P_0P_{10}}$, then it calculates the distance of each point in $\{P_0, \ldots, P_{10}\}$ to $\overrightarrow{P_0P_{10}}$. It finds that P_4 has the maximum distance to $\overrightarrow{P_0P_{10}}$, which is greater than ϵ. Then it goes to compress sub-trajectories $[P_0, \ldots, P_4]$ and $[P_4, \ldots, P_{10}]$, separately. When using SED (right), the sub-trajectory $[P_4, \ldots, P_{10}]$ is further split to $[P_4, \ldots, P_7]$ 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. □

*4.1.2 Algorithm Theo-Pavlidis (*TP*) [46].* It originally employs the global checking policy to output disjoint line segments, and we slightly modify it to have continuous line segments.

Given a trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ and an error bound ϵ, algorithm TP begins by creating the finest possible trajectory approximation: $[P_0, P_1], [P_1, P_2], \ldots, [P_{n-1}, P_n]$, 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 $ped(P_{s+i}, \overrightarrow{P_sP_{s+k}})$ 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 $ped(P_1, \overrightarrow{P_0P_2}) = 0.32\epsilon$. (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) $\overrightarrow{P_6P_8}$ and $\overrightarrow{P_8P_9}$ are merged to $\overrightarrow{P_6P_9}$. The cost of merging $\overrightarrow{P_5P_6}$ and $\overrightarrow{P_6P_9}$, and the cost of merging $\overrightarrow{P_6P_9}$ and $\overrightarrow{P_9P_{10}}$ are also updated. (4) At last, the algorithm outputs two line segments $\overrightarrow{P_0P_4}$ and $\overrightarrow{P_4P_{10}}$. □

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 [39] is a combination of top-down algorithm DP and opening windows while SWAB [25] 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 [31]. Besides, in environments like wireless sensor networks, online algorithms also need to address the problem of balancing the trade-off between the energy cost due to communication and the accuracy of the trajectories' detection and representation [16], and to control the "freshness" (i.e., the latency) of the simplified data by a careful management of the data buffer of an online algorithm [16]. To design more efficient online algorithms, techniques typically need to be designed closely coupled with distance metrics. Indeed, BQS [31] and SQUISH-E [43] propose to utilize convex hulls and priority queues, respectively, and they speed up trajectory simplification using PED and SED, respectively. DOTS [4] is an online method using LISSED that could be computed in an incremental way. To our knowledge, no specific techniques have been developed for DAD. Hence, *we choose algorithms* BQS *(the performance optimized online algorithm, specific for* PED*),* SQUISH-E *(the famous online algorithm, specific for* SED*),* DOTS *(using* LISSED *and recommended in the recent evaluation work [66]) and* OPW *(a well-known online algorithm based on* DP *that is compatible with* DAD*) as the representatives of online algorithms using* PED*,* SED*,* LISSED *and* DAD*, respectively.*

4.2.1 Algorithm OPW *[39].* 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 $\ddot{T}[P_0, \ldots, P_n]$ and an error bound ϵ, algorithm OPW [39] maintains a window $W[P_s, \ldots, P_k]$, 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 $W[P_s, \ldots, P_k]$ 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 $[P_s, \ldots, P_k, P_{k+1}]$ ($k + 1 \leq n$) 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 $[P_{k-1}, \ldots, P_{k+1}]$. The above process repeats until all points in $\ddot{T}$ have been considered. The process is similar for SED and DAD. The time complexity of algorithm OPW remains in $O(n^2)$ time, the same as the DP algorithm.

4.2.2 Algorithm BQS Using PED *[31].* It is essentially an efficiency optimized OPW algorithm [39], and reduces the running time by introducing convex hulls to pick out a certain number of points, which makes it dedicated for PED.

For a buffer W with sub-trajectory $[P_s, \ldots, P_k]$, it splits the space into four quadrants. A buffer here is similar to a window in OPW [39]. For each quadrant, a rectangular bounding box is firstly created using the least and highest x and y values among points $\{P_s, \ldots, P_k\}$, 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 $\overrightarrow{P_sP_h}$ and $\overrightarrow{P_sP_l}$ have the largest and smallest angles with the x-axis, respectively. Here $P_h, P_l \in \{P_s, \ldots, P_k\}$. 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 $\overrightarrow{P_sP_k}$, 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 $[P_s, \ldots, P_k]$ to line $\overrightarrow{P_sP_k}$. (1) If $d_l \geq \epsilon$, it produces a new line segment $\mathcal{L}(P_s, P_{k-1})$, and produces a new window $[P_{k-1}, \ldots, P_k]$ to replace W. (2) If $d_u < \epsilon$, it

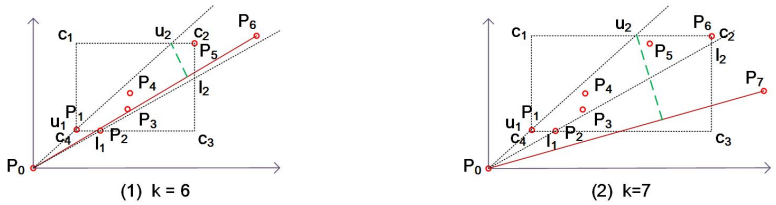

Fig. 4. Examples for algorithm BQS.

simply expands buffer W to $[P_s, \ldots, P_k, P_{k+1}]$ $(k+1 \leq n)$ 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 $\overrightarrow{P_sP_h} = \overrightarrow{P_0P_1}$ and $\overrightarrow{P_sP_l} = \overrightarrow{P_0P_2}$ 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 $\overrightarrow{P_0P_6}$ when $k = 6$ or to line $\overrightarrow{P_0P_7}$ when $k = 7$. When $k = 6$, all these distances to $\overrightarrow{P_0P_6}$ are less than ϵ, hence BQS goes on to the next point (case 2); When $k = 7$, the max and min distances to $\overrightarrow{P_0P_7}$ are larger and less than ϵ, respectively, and BQS needs to compress sub-trajectory $[P_0, \ldots, P_7]$ along the same line as DP (case 3). □

4.2.3 Algorithm SQUISH-E Using SED *[43].* It is an online bottom-up algorithm that is dedicated 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 $P(pre, suc, mnprio, prio)$, 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 $sed(P_i, \overrightarrow{pre(P_i)suc(P_i)})$. Then, SQUISH-E finds and removes a point P_j from Q that has the lowest priority $prio(P_j) < \epsilon$, and the properties $mnprio$ of predecessor $pre(P_j)$ and successor $suc(P_j)$ are updated to $\max(mnprio(pre(P_j)), prio(P_j))$ and $\max(mnprio(suc(P_j)), prio(P_j))$, respectively. Next, the properties $prio$ of $pre(P_j)$ and $suc(P_j)$ are further updated to $mnprio(pre(P_j)) + sed(pre(P_j), \overrightarrow{pre(pre(P_j))suc(P_j)})$ and $mnprio(suc(P_j)) + sed(suc(P_j), \overrightarrow{pre(P_j)suc(suc(P_j))})$, 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 $(pre, suc, mmprio, prio)$ 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_1 has the minimal value, thus, it is removed from the list. The $mnprio$ properties of P_0 and P_2 are updated to $max\{mnprio(pre(P_1)), prio(P_1)\} = max\{mnprio(P_0), prio(P_1)\} = max\{0, 0.42\epsilon\} = 0.42\epsilon$, and

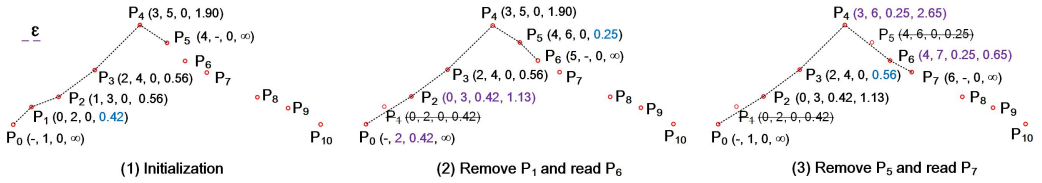

Fig. 5. The trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ is compressed by the SQUISH-E algorithm using SED to five line segments. The size of Q is 6, and the data structure after point P is a tuple (pre, suc, $mmprio$, $prio$).

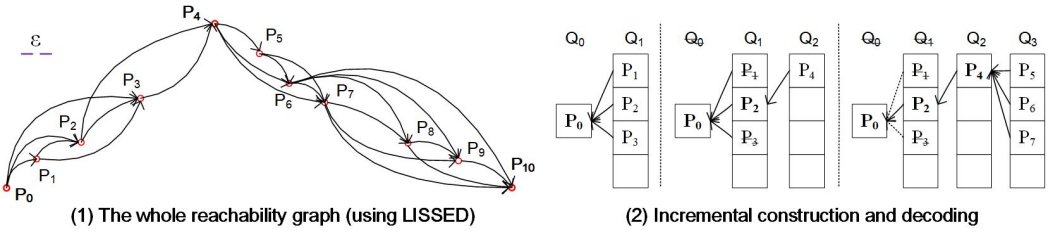

Fig. 6. Example of algorithm DOTS using LISSED with error bound of ϵ^2, where (1) left is the supposed reachability graph of trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$, and (2) right is the incremental construction of reachability graph and online decoding of the shortest path. Finally, the trajectory is compressed to four line segments.

$max\{mnprio(P_2),\ prio(P_1)\} = 0.42\epsilon$, respectively. Furthermore, the $prio$ property of P_2 is updated to $mnprio(suc(P_j)) + sed(suc(P_j), \overrightarrow{pre(P_j)suc(suc(P_j))}) = mnprio(P_2) + sed(P_2, \overrightarrow{P_0P_3}) = 0.42\epsilon + 0.71\epsilon = 1.13\epsilon$, and the $prio$ property of P_0 is still ∞. 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 $\overrightarrow{P_9P_{10}}$. □

4.2.4 Algorithm DOTS Using LISSED *[4].* It is a directed acyclic graph based online trajectory simplification method that supports LISSED. Other than the optimal and near optimal algorithms [8, 11] that find a shortest path after completely constructing the reachability graph, DOTS incrementally determines a shortest path when constructing the reachability graph [4].

Algorithm DOTS first shows that the reachability graph of an input trajectory retrogrades to a tree structure. It also observes that the local approximation error $lissed(P_s, P_{s+k})$ is growing with respect to k, thus, it is reasonable to assume that k should not be too large. By this assumption, each layer of the tree could be added without feeding the entire trajectory, and the shortest path could also be determined recursively, *i.e.,* layer by layer. For each node (point) in each layer, DOTS marks those having descendants as *alive* and others as *dead*. Initially, the first layer Q_0 is $\{P_0\}$, and it is *alive*. Then, let Q_c ($c > 0$) be the current layer under construction, DOTS in turn adds point P_{s+k}, which has an LISSED distance to any point of Q_{c-1} less than the error bound, to Q_c, until no points can be added to Q_c (recall that k should not be too large). After that, it updates the status of each layer that has *alive* points: the parent of an *alive* point may be adjusted to minimize the total integral square SED of the path, points without descendants are marked as *dead*, and those 1-alive-element layers are decoded and their *alive* points are output to the simplified trajectory. The time complexity of DOTS is $O(n^2/K)$, where K is the number of output points, and the space complexity of DOTS is $O(|Q|^2)$, where $|Q|$ is the size of a layer.

Example 7: Figure 6 is an example of DOTS. The reachability graph is constructed layer by layer incrementally, and a part of the shortest path is determined before the whole graph is completely

constructed. (a) Initially, P_0 is added to the first layer Q_0, and it is the only element in the layer. (b) Then, points P_1, P_2 and P_3 are in turn added to the second layer Q_1. Since P_0 does not have any new child point, DOTS updates the status of Q_1 and outputs P_2 as it is the only *alive* point in Q_1. (c) The process repeats. Point P_4 is added to layer Q_2, points P_5, P_6 and P_7 are added to layer Q_3, and so on. (d) Finally, it outputs 4 line segments $\overrightarrow{P_0P_2}$, $\overrightarrow{P_2P_4}$, $\overrightarrow{P_4P_7}$ and $\overrightarrow{P_7P_{10}}$. □

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) [51] 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 [30] 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 [13, 55, 63, 68]. 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 [59] follows the similar routine as RW except that it uses SED and assumes a velocity $\vec{v}$ for each section. [59] proves that if LDR uses $\epsilon/2$ as the threshold in position tracking, then its output trajectory has a max error not greater than ϵ to the original trajectory. Thus, it can be treated as a trajectory simplification algorithm as well. It has poor compression ratios because both the value and the direction of velocity $\vec{v}$ are pre-defined and fixed between two updates, and it indeed uses a half ϵ. Recently algorithm CISED [29] 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, and *we choose* Intersect *[33] and* Interval *[23] as the representatives of one-pass algorithms using* DAD.

Algorithms SIPED, LDR, OPERB, CISED and Intersect share a common idea, *i.e.,* using a half-ϵ to implement the strong simplification to ensure that the max error does not exceed ϵ. In the sequel, we shall discuss that the half-ϵ of SIPED and CISED can be extended to the full-ϵ with some small modifications, along the similar way that Interval extends Intersect.

4.3.1 Algorithm OPERB *Using* PED *[30].* It designs a local distance checking method to dynamically adjust the direction of line segments to achieve an effective one-pass process. It has strong and weak versions, called OPERB and OPERB-A, respectively. OPERB-A shares the same routine as OPERB except that it applies a lazy output policy and allows data interpolations.

Consider an error bound ϵ and a sub-trajectory $\ddot{\mathcal{T}}_s[P_s, \ldots, P_{s+k}]$. 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 $\{P_s, \ldots, P_{s+i}\}$) to *fit* all the previously processed points $\{P_s, \ldots, P_{s+i}\}$. 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 $\{P_s, \ldots, P_{s+i}\}$ to $\mathcal{R}_{i+1} = \overrightarrow{P_sP_{s+i+1}}$ 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.

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

where (a) $1 \leq i \leq k+1$; (b) $\mathcal{R}_{i-1} = \overrightarrow{P_sP_{s+i-1}}$, is the directed line segment whose end point P_{s+i-1} is in $\ddot{\mathcal{T}}_s[P_s, \ldots, P_{s+k}]$; (c) $\mathcal{L}_i$ is the directed line segment built by fitting function $\mathbb{F}$ to fit sub-trajectory $\ddot{\mathcal{T}}_s[P_s, \ldots, P_{s+i}]$ 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 $(-2\pi, -\frac{3\pi}{2}]$, $[-\pi, -\frac{\pi}{2}]$, $[0, \frac{\pi}{2}]$ 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 8: Figure 7 is a running example of the OPERB algorithm compressing the same trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$. (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 $\overrightarrow{P_4P_{10}}$. □

4.3.2 Algorithm SIPED *Using* PED *[13, 55, 63, 68].* It develops a concept of "Sector" [13, 55, 63, 68], which converts the distance tolerance into the angle change tolerance for checking points.

Given a sequence of points $[P_s, P_{s+1}, \ldots, P_{s+k}]$ 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 $\overrightarrow{P_sP^u_{s+i}}$ and $\overrightarrow{P_sP^l_{s+i}}$ such that $ped(P_{s+i}, \overrightarrow{P_sP^u_{s+i}}) = ped(P_{s+i}, \overrightarrow{P_sP^l_{s+i}}) = \epsilon$ and either ($\overrightarrow{P_sP^l_{s+i}}.\theta < \overrightarrow{P_sP^u_{s+i}}.\theta$ *and* $\overrightarrow{P_sP^u_{s+i}}.\theta - \overrightarrow{P_sP^l_{s+i}}.\theta < \pi$) or ($\overrightarrow{P_sP^l_{s+i}}.\theta > \overrightarrow{P_sP^u_{s+i}}.\theta$ *and* $\overrightarrow{P_sP^u_{s+i}}.\theta - \overrightarrow{P_sP^l_{s+i}}.\theta < -\pi$). Indeed, they form a *sector* $\mathcal{S}(P_s, P_{s+i}, \epsilon)$ that takes P_s as the center point and $\overrightarrow{P_sP^u_{s+i}}$ and $\overrightarrow{P_sP^l_{s+i}}$ as the borderlines. There exists a data point Q such that for any data point P_{s+i} ($i \in [1, \ldots k]$), 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 $\mathcal{S}(P_s, P_{s+i}, \epsilon)$ ($i \in [1, k]$) share common data points other than P_s, *i.e.,* $\bigcap_{i=1}^{k}\mathcal{S}(P_s, P_{s+i}, \epsilon) \neq \{P_s\}$ [55, 63, 68]. Here, point Q may not belong to $\{P_s, P_{s+1}, \ldots, P_{s+k}\}$. 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} ($j \in [1, \ldots i]$), its PED to line segment $\overline{P_sP_{s+i}}$ is not greater than the error bound ϵ if $\bigcap_{j=1}^{i}\mathcal{S}(P_s, P_{s+j}, \epsilon/2) \neq \{P_s\}$, as pointed out in [68]. 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 $[P_s, P_{s+1}, \ldots, P_{s+k}]$ 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 ϵ-$\mathcal{S}$ by adding a constraint. That is, for any point P_{s+j} ($j \in [1, \ldots i]$), its PED to line segment $\overline{P_sP_{s+i}}$ is not greater than the error bound ϵ if $P_{s+i} \neq P_s$ and $P_{s+i} \in \bigcap_{j=1}^{i-1}\mathcal{S}(P_s, P_{s+j}, \epsilon)$, *i.e., P_{s+i}* lives in the common intersection of the preview full *sectors.*

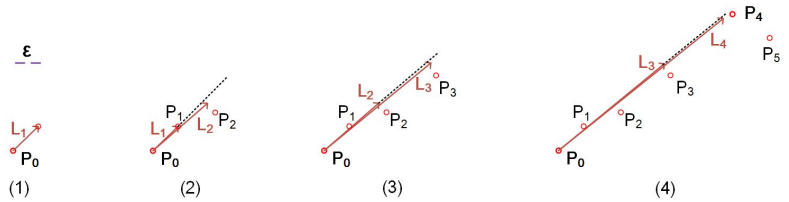

Fig. 7. The trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ is compressed by the OPERB algorithm using PED to two line segments.

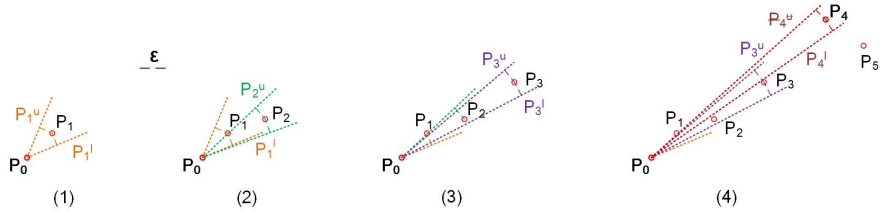

Fig. 8. The trajectory $\ddot{\mathcal{T}}$ is compressed by the sector intersection algorithm using PED to two line segments.

Example 9: Figure 8 is a running example of algorithm SIPED ($\frac{\epsilon}{2}$) taking as input the same trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$. 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* $\mathcal{S}(P_0, P_3, \epsilon/2)$ takes P_0 as the center point and $\overrightarrow{P_0P_3^u}$ and $\overrightarrow{P_0P_3^l}$ as the borderlines. Because $\bigcap_{i=1}^{4} \mathcal{S}(P_0, P_{0+i}, \epsilon/2) \neq \{P_0\}$ and $\bigcap_{i=1}^{5} \mathcal{S}(P_0, P_{0+i}, \epsilon/2) = \{P_0\}$, $\overrightarrow{P_0P_4}$ 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 $\overrightarrow{P_4P_{10}}$. □

4.3.3 Algorithm CISED *Using* SED *[29].* It develops an idea of *spatio-temporal cone* that extends the *sector intersection* method [13, 55, 63, 68] from an *x-y* 2D space to an *x-y-t* spatio-temporal 3D space. Note that for computer graphics and cartography, there is another 3D space, *i.e., x-y-z* 3D space, where *z* is the height. There are studies [1, 15] for solving the *min-#* and *min-ϵ* problems in the *x-y-z* 3D space, *e.g.,* [1] extends the *sector intersection* method to the *off-line ball-inclusion testing* in the *x-y-z* 3D space, so as to develop efficient near quadratic time algorithms.

Given a sub-trajectory $[P_s, ..., P_{s+k}]$ 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 $\ddot{\mathcal{T}}_s$ *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 $\{P_s, P_{s+1}, \ldots, P_{s+k}\}$. If point Q is not necessarily in $\{P_s, P_{s+1}, \ldots, P_{s+k}\}$, then this algorithm is a weak simplification, named CISED-W that uses a full-ϵ cone. If point Q must be P_{s+i} $(1 \leq i \leq k)$, then for any point P_{s+j} $(j \in [1, ...i])$, its SED to line segment $\overline{P_sP_{s+i}}$ is not greater than the error bound ϵ if $\bigcap_{j=1}^{i} C(P_s, P_{s+j}, \epsilon/2) \neq \{P_s\}$ as pointed out in [29]. This algorithm, named CISED ($\frac{\epsilon}{2}$) that uses a half-ϵ cone, belongs to strong simplification. Moreover, CISED ($\frac{\epsilon}{2}$) can also be extended to CISED (ϵ), another strong version that uses a full-ϵ cone, by adding a constraint that P_{s+i} lives in the common intersection of the preview full *cones*.

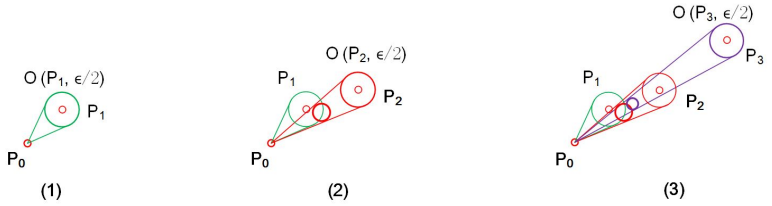

Fig. 9. A running example of the CISED algorithm. The points and the oblique circular cones are projected on an x-y space.

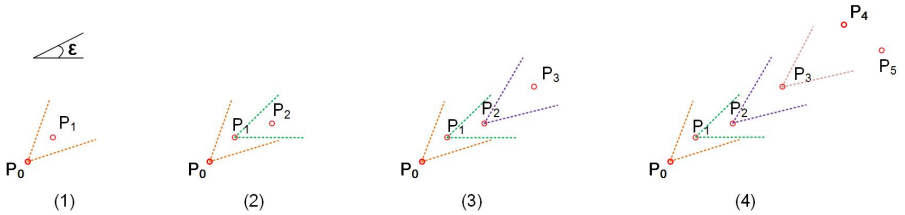

Fig. 10. The trajectory $\ddot{\mathcal{T}}$ is compressed by the interval algorithm using DAD to two line segments.

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.

Example 10: Figure 9 shows a running example of algorithm CISED ($\frac{\epsilon}{2}$) for compressing the trajectory $\ddot{\mathcal{T}}$ 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 $P.t - P_1.t = 0$ 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 $\mathcal{R}_1 \bigcap \mathcal{R}_2 \neq \emptyset$. (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.,* $\{\overrightarrow{P_0P_2}, \overrightarrow{P_2P_4}, \overrightarrow{P_4P_7}, \overrightarrow{P_7P_{10}}\}$. □

4.3.4 Algorithms Intersect *[33] and* Interval *[23] 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 [13, 55, 63, 68].

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 [33]. The common intersection of *direction ranges* of directed line segments $\{\overrightarrow{P_sP_{s+1}}, \overrightarrow{P_{s+1}P_{s+2}}, ..., \overrightarrow{P_{s+k-1}P_{s+k}}\}$ *w.r.t.* ϵ is $\bigcap_{i=1}^{k} Range(\overrightarrow{P_{s+i-1}P_{s+i}}.\theta, \epsilon)$.

Algorithm Intersect [33] 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 angle 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 [23] 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

$\overrightarrow{P_sP_{s+k}}.\theta$ falls in the common intersection, then the angle 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 11: Figure 10 is a running example of the *interval* method taking as input the same trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$. At the beginning, P_0 is the first start point, and points P_1, P_2, P_3, etc., each has a *direction range* $Range(\overrightarrow{P_0P_1}, \epsilon)$, $Range(\overrightarrow{P_1P_2}, \epsilon)$, $Range(\overrightarrow{P_2P_3}, \epsilon)$, etc., respectively. Because $\bigcap_{i=1}^{4} Range(\overrightarrow{P_{0+i-1}P_{0+i}}, \epsilon) \neq \phi$ and $\overrightarrow{P_0P_4}.\theta$ falls in the *common sub-interval*, 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 $\overrightarrow{P_4P_{10}}$. □

5 TRAJECTORY AGING

Suppose that we have compressed a trajectory $\ddot{\mathcal{T}}_0$ to $\overline{\mathcal{T}}_1$ using any LS algorithm $\mathcal{A}$ with an error bound ϵ_1. As time evolves, we may need to further compress $\overline{\mathcal{T}}_1$ to an even coarser trajectory $\overline{\mathcal{T}}_2$. What are the relationships among $\overline{\mathcal{T}}_2$, $\overline{\mathcal{T}}_1$ and $\ddot{\mathcal{T}}_0$? And what is the right way to get the coarser trajectory $\overline{\mathcal{T}}_2$? What line simplification algorithms should we use in the first and second rounds of simplifications? If the first round of simplification uses algorithm $\mathcal{A}_1$, can we use algorithm $\mathcal{A}_2$ in the second round? How to set the parameter of error bounds in these simplifications? And after multiple rounds of simplifications, does the coarse trajectory still have bounded errors *w.r.t.* the original trajectory? This section is to answer these questions from the views of friendliness [3] (Section 5.1) and errors (Section 5.2). Note that (1) if we get $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ by optimal algorithms *w.r.t.* error bounds ϵ_1 and ϵ_2, respectively, then $\overline{\mathcal{T}}_2$ may not be the optimal one of $\ddot{\mathcal{T}}_0$ *w.r.t.* any error bound [3], and (2) as LISSED is a special case of SED, we refer to the SED errors rather than the LISSED errors when we talk about aging friendliness and errors of algorithms using LISSED (recall that *if the* LISSED *error bound of such an algorithm is set to* ϵ^2*, then its maximal* SED *error is not greater than* ϵ). The major results are summarized in Table 3.

5.1 Friendliness of Data Aging

We first discuss the relationship between $\overline{\mathcal{T}}_1$ and $\overline{\mathcal{T}}_2$ from the view of friendliness, which was defined in [3], but was seldom discussed later.

Aging friendly [3]. 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 [3] 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 [3], 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 11, where error bounds $\epsilon_1 = 30^o$ and $\epsilon_2 = 45^o$.

(1) Optimal($\ddot{\mathcal{T}}$, 45^o, 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 $\{P_0, P_2, P_3\}$.

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	× ([3])	$\epsilon_1 + \epsilon_2$ (Prop. 5.6)
	DAD	× (Prop. 5.1)	$\epsilon_1 + \epsilon_2$ (Prop. 5.6)
Batch algorithm DP	PED or SED	✓([3])	$\max(\epsilon_1, \epsilon_2)$ (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. 11. 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 $\epsilon_2 = 45^o$ to three points $\{P_0, P_2, P_3\}$, and (2) it is first compressed using $\epsilon_1 = 30^o$ to three points $\{P_0, P_2, P_3\}$, then compressed using $\epsilon_2 = 45^o$ to two points $\{P_0, P_3\}$.

(2) Optimal(Optimal($\ddot{\mathcal{T}}$, 30^o, DAD), 45^o, DAD). In the first round ($\epsilon_1 = 30^o$), the original trajectory is compressed to three points $\{P_0, P_2, P_3\}$, and in the second round ($\epsilon_2 = 45^o$), 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^o, it is finally compressed to two points $\{P_0, P_3\}$.

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

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 12, where error bounds $\epsilon_1 = 30^o$ and $\epsilon_2 = 45^o$.

(1) DP($\ddot{\mathcal{T}}$, 45^o, 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^o, 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^o, DAD), 45^o, DAD). In the first round ($\epsilon_1 = 30^o$), the original trajectory is compressed to three points $\{P_0, P_2, P_4\}$, and in the second round ($\epsilon_2 = 45^o$), 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^o, it is finally compressed to two points $\{P_0, P_4\}$.

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

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 12-(2), the angular deviations of line segments $\overline{P_1P_2}$ and $\overline{P_0P_2}$ 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 ($\epsilon_1 = 30^o$), the point P_2 serves as a splitting point while in the second round of run ($\epsilon_2 = 45^o$), 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.

Fig. 12. A counter example of the aging friendliness of DP using DAD, where (1) the original trajectory is compressed using $\epsilon_2 = 45^o$ to three points $\{P_0, P_2, P_4\}$, and (2) the original trajectory is first compressed using $\epsilon_1 = 30^o$ to three points $\{P_0, P_2, P_4\}$, then compressed using $\epsilon_2 = 45^o$ to two points $\{P_0, P_4\}$.

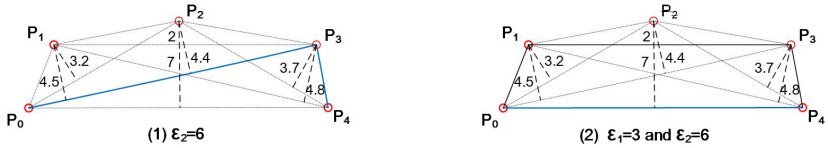

Fig. 13. A counter example of the aging friendliness of algorithm TP, where (1) the original trajectory is compressed using $\epsilon_2 = 6$ to three points $\{P_0, P_3, P_4\}$, and (2) the original trajectory is first compressed using $\epsilon_1 = 3$ to four points $\{P_0, P_1, P_3, P_4\}$, then compressed using $\epsilon_2 = 6$ to two points $\{P_0, P_4\}$.

Proof: We shall prove this by constructing a counter-example shown in Figure 13, 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 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($\ddot{\mathcal{T}}$, 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, TP($\ddot{\mathcal{T}}$, 6, PED) $\neq$ TP(TP($\ddot{\mathcal{T}}$, 3, PED), 6, 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. □

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) $\mathcal{A}(\ddot{\mathcal{T}}, \epsilon_2, \mathcal{M})$. As shown in Figure 14-(1)(3)(5), the algorithm $\mathcal{A}$ incrementally reads $\{P_0, P_1, \ddot{P}_5\}$ 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 $\{P_0, P_5, P_6\}$.

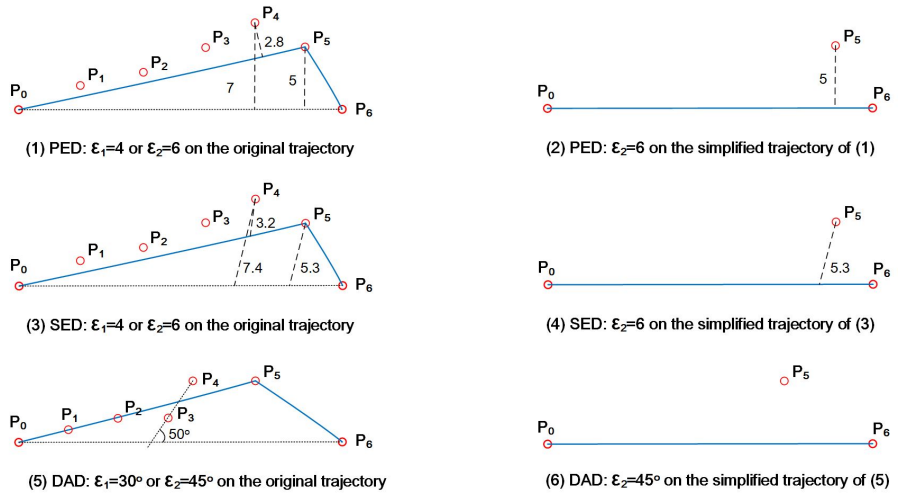

Fig. 14. Counter examples of the aging friendliness of incremental algorithms (either online or one-pass).

(2) $\mathcal{A}(\mathcal{A}(\dddot{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$. When using $\epsilon_1 = 4$ or $\epsilon_1 = 30^o$, the algorithm also outputs $\{P_0, P_5, P_6\}$. Then $\{P_0, P_5, P_6\}$ is compressed using $\epsilon_2 = 6$ or $\epsilon_2 = 45^o$ to $\{P_0, P_6\}$, as shown in Figure 14-(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. □

Though DP (using either PED or SED) is the only algorithm with the aging friendly feature, it is not necessarily the only algorithm that we have to use for compressing trajectories. Indeed, all the other algorithms are also applicable to compress trajectories in data aging although these algorithms may lead to (a bit) poorer compression ratios compared with DP. However, compression ratios are only one aspect of qualities for line simplification algorithms. Further, to keep aging friendliness, each run of algorithm DP must take as input the entire original or simplified trajectory that has the same start and end data points, and is not aging friendly, otherwise.

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 $\overline{\dddot{\mathcal{T}}}_2$ and the original trajectory $\dddot{\mathcal{T}}_0$.

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 the original trajectory $\dddot{\mathcal{T}}$ and simplified trajectory $\overline{\dddot{\mathcal{T}}} = DP(DP(\dddot{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$ 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 [3], $DP(DP(\dddot{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}) = DP(\dddot{\mathcal{T}}, \epsilon_2, \mathcal{M})$, which has the max error of ϵ_2 to the original trajectory $\dddot{\mathcal{T}}$.

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

- For a trajectory $\dddot{\mathcal{T}}$ 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 $\dddot{\mathcal{T}} = [P_0, P_1, P_2]$ ($n = 3$), if the distance from P_1 to $\overline{P_0P_2}$ is less than ϵ_1, then $DP(\dddot{\mathcal{T}}, \epsilon_1, \mathcal{M}) = [P_0, P_2]$. Obviously

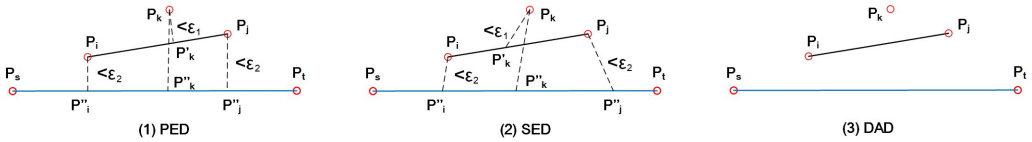

Fig. 15. Examples of aging errors.

$DP(DP(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}))$ is $[P_0, P_2]$ too; if the distance from P_1 to $\overline{P_0P_2}$ is larger than ϵ_1, then $DP(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M}) = \ddot{\mathcal{T}}$, and $DP(DP(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}) = DP(\ddot{\mathcal{T}}, \epsilon_2, \mathcal{M}) = \ddot{\mathcal{T}}$.

- Assume that it is true for every trajectory $\ddot{\mathcal{T}}$ 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 $DP(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M}) = [P_0, P_n]$, and $DP(DP(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}) = DP([P_0, P_n], \epsilon_2, \mathcal{M}) = [P_0, P_n]$. If $d_{max} > \epsilon_1$, then in $DP(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M})$, point P_i will split the trajectory $\ddot{\mathcal{T}}$ into two sub-trajectories, *i.e.,* $[P_0, ..., P_i]$ and $[P_i, ..., P_n]$, and continue to simplify each sub-trajectory. Hence, the result of $DP(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M})$ is the union of $DP([P_0, ..., P_i], \epsilon_1, \mathcal{M})$ and $DP([P_i, ..., P_n], \epsilon_1, \mathcal{M})$. Obviously, the points P_0, P_i and P_n are in the simplified trajectory of $DP(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M})$, and P_i is still the first splitting point of the DP algorithm taking the simplified trajectory and ϵ_2 as input. Hence, $DP(DP(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$ is the union of $DP(DP([P_0, ..., P_i], \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$ and $DP(DP([P_i, ..., P_n], \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$. By the assumption, we have $DP(DP([P_0, ..., P_i], \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}) = DP([P_0, ..., P_i], \epsilon_1, \mathcal{M})$ and $DP(DP([P_i, ..., P_n], \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}) = DP([P_i, ..., P_n], \epsilon_1, \mathcal{M})$. Thus, $DP(DP(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}) = DP(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M})$.
- Now we have $DP(DP(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M}) = DP(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M})$, whose max error to the original trajectory $\ddot{\mathcal{T}}$ is ϵ_1.

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

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 the original trajectory $\ddot{\mathcal{T}}$ and simplified trajectory $\overline{\mathcal{T}} = \mathcal{A}(\mathcal{A}(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$ 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 15). 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 to $\overline{P_sP_t}$, *i.e.,* $|P_kP''_k|$, is less than $\epsilon_1 + \epsilon_2$. If it is DAD, then obviously the error between $\overline{P_kP_{k+1}}$ and $\overline{P_sP_t}$ 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 Figure 12-(2) we can find that the error from $\overline{P_3P_4}$ to $\overline{P_0P_4}$ is $\angle P_3P_4P_0 = \angle P_3P_4P_2 + \angle P_2P_4P_0$, whose bound is not less than $\epsilon_1 + \epsilon_2$, thus the error bound between the original trajectory $\ddot{\mathcal{T}}$ and simplified trajectory $\overline{\mathcal{T}} = \mathcal{A}(\mathcal{A}(\ddot{\mathcal{T}}, \epsilon_1, \mathcal{M}), \epsilon_2, \mathcal{M})$ is not less than $\epsilon_1 + \epsilon_2$. If $\mathcal{A}$ is a bottom-up or incremental algorithm, either online or one-pass, then from Figure 13-(2) or Figure 14 we also have the conclusion.

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

Table 4. Real-life trajectory datasets

Data sets	Number of Trajectories	Sampling Rates (s)	Points Per Trajectory	Total Points
UCar [29]	1000	3-5	~ 114.0K	114.0M
Geolife [69]	182	1-5	~ 131.4K	24.2M
Mopsi [36]	51	2	~ 153.9K	7.9M

By Propositions 5.5 and 5.6, it is obvious that all the above algorithms have bounded errors in data aging, which make us freely re-compress trajectories by any of these algorithms as long as we use the same distance metric.

6 EVALUATION

In this section, we present extensive and systematic experimental studies and analyses of twelve 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) [29], Geolife trajectory data (Geolife) [69] and Mopsi trajectory data (Mopsi) [36], 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 time interval between two neighbouring data points is occasionally very long, *e.g.,* greater than 10^7 seconds. The data source and sampling rate also affect the performance of LS algorithms using certain distance metrics.
(1) UCar (also called ServiceCar in [29]) 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 [69] 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 [36] 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, SQUISH-E and DOTS, 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 ($\frac{\epsilon}{2}$). Besides, algorithms OPERB [30] and CISED [29] both have weak versions, named OPERB-A and CISED-W, respectively. For algorithm CISED, we fixed parameter $m = 16$ as evaluated in [29], *i.e.,* 16-edges inscribe regular polygon. For algorithm OPERB, we remove its fifth optimization technique to make it fit for the definition of the tolerance-zone error measure [1, 6, 11, 22, 38] (otherwise, it is the *infinite beam* error measure [6, 11]). 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^o to 90^o 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 (*i.e.,* from the first 1,000 points to the first 10,000 points of the trajectory) 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 measures to evaluate the aging friendliness of LS algorithms. Besides, the running time is used to evaluate the efficiency of LS algorithms.

Compression ratios. For trajectories $\{\ddot{\mathcal{T}}_1, \ldots, \ddot{\mathcal{T}}_M\}$ and their piece-wise line representations $\{\overline{\mathcal{T}_1}, \ldots, \overline{\mathcal{T}_M}\}$, 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.

Simplification errors. The max (respectively average) simplification error is the max (respectively average) value of the distances from every point of the original trajectories to its representing line segment of the simplified trajectories.

Running time. It is the efficiency of algorithms.

6.3 Experimental Results and Analyses

We next present our findings.

6.3.1 Evaluation and Analysis of Compression Ratio. The compression ratios of these algorithms under varied error bounds ϵ and trajectory sizes are reported in Figures 16, 17, 18, 19, 20 and 21. Note that the optimal algorithm using SED and DAD is not reported in Figures 16, 17 and 18 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. When using DAD, the dataset collected by cars (*e.g.,* UCar) has better compression ratios than the datasets partially collected by individuals (*e.g.,* Geolife and Mopsi), as cars typically move more regularly than individuals in directions.

(2) Compression ratios are insensitive to the sizes of trajectories. The reason lies in that once the distance is greater than the error bound, a line segment is produced, and the compression continues to repeat this process, which is essentially irrelevant to the sizes of trajectories. This is also the key to preserve the error bound for any of the optimal, batch, online and one-pass algorithms.

(3) The compression ratios of algorithms using PED from the best to the worst are normally the optimal algorithm Optimal, online algorithm BQS, one-pass algorithm SIPED (ϵ), batch algorithms TP and DP and one-pass algorithm OPERB-A, 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, DP and OPERB-A are comparable, and their output sizes are on average (125.05%, 131.01%, 138.01%), (130.03%, 140.56%, 139.00%) and (134.16%, 137.73%, 144.31%) 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, OPERB-A) are (1.6%, 2.2%, 2.2%, 1.9%, 2.0%, 2.4%, 2.4%, 2.3%) when $\epsilon = 40m$.

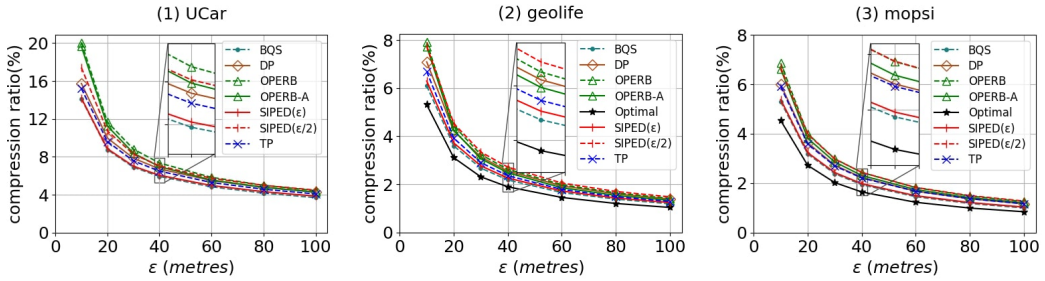

Fig. 16. Evaluation of compression ratios (PED) on full datasets: varying the error bound ϵ.

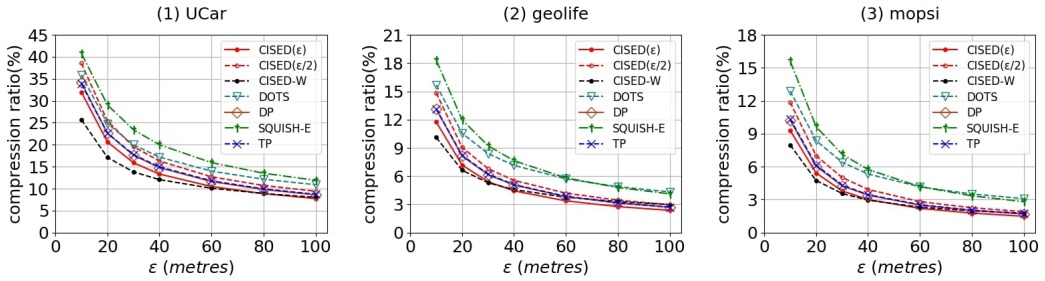

Fig. 17. Evaluation of compression ratios (SED) on full datasets: varying the error bound ϵ.

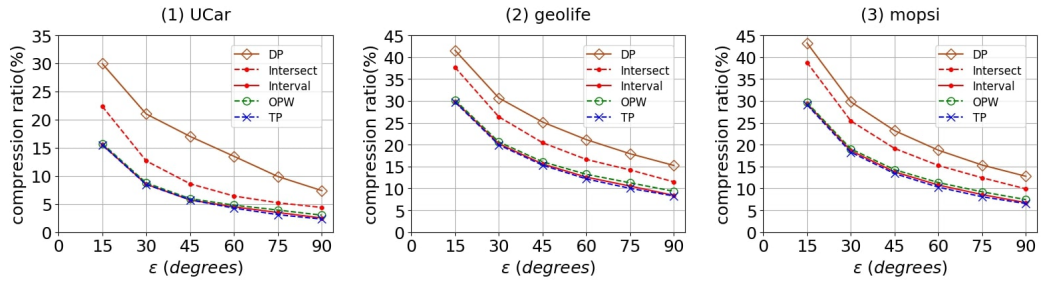

Fig. 18. Evaluation of compression ratios (DAD) on full datasets: varying the error bound ϵ.

(4) The compression ratios of algorithms using SED from the best to the worst are normally the Optimal algorithm, one-pass algorithms CISED-W and CISED (ϵ), batch algorithms TP and DP, one-pass algorithm CISED ($\frac{\epsilon}{2}$), and online algorithm DOTS and 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-W, CISED (ϵ), CISED ($\frac{\epsilon}{2}$), SQUISH-E and DOTS are on average (100.98%, 108.16%, 110.15%), (109.27%, 110.13%, 115.90%,), (134.35%, 159.30%, 136.06%), (165.94%, 225.68%, 206.90%) and (140.98%, 200.36%, 198.73%) of Optimal on (UCar, Geolife, Mopsi), respectively. For example, in Mopsi, the compression ratios of algorithms (TP, DP, SQUISH-E, DOTS, CISED (ϵ), CISED ($\frac{\epsilon}{2}$), CISED-W) are (3.45%, 3.41%, 5.75%, 5.34%, 3.02%, 3.86%, 2.96%), respectively, when $\epsilon = 40m$.

(5) 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 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%)

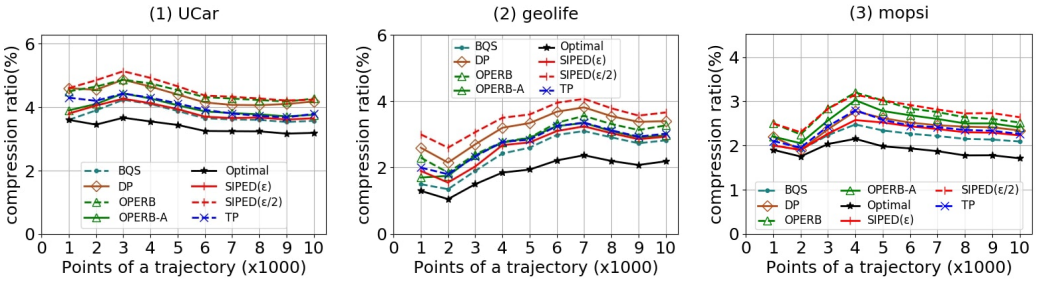

Fig. 19. Evaluation of compression ratios (PED) on small datasets: varying the size of trajectories.

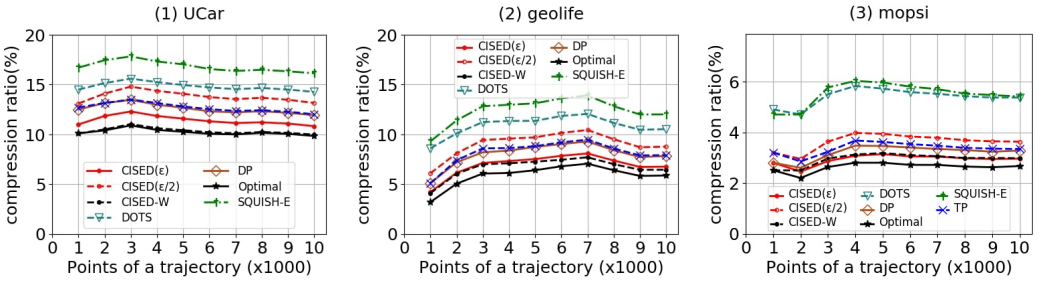

Fig. 20. Evaluation of compression ratios (SED) on small datasets: varying the size of trajectories.

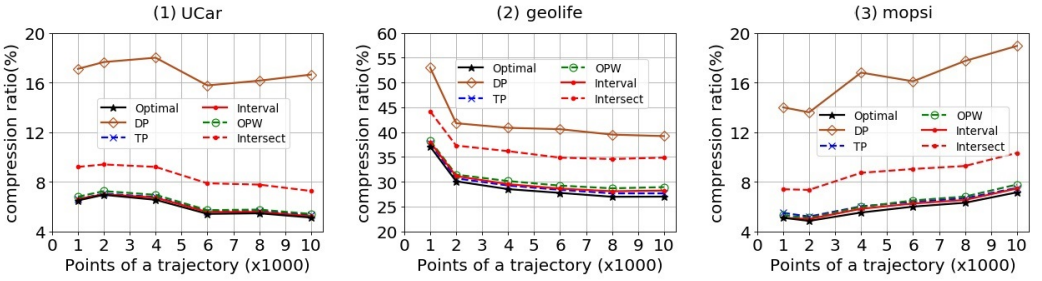

Fig. 21. Evaluation of compression ratios (DAD) on small datasets: varying the size of trajectories.

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 ϵ = 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 $[P_s, ..., P_e]$ 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 $\overrightarrow{P_sP_e}$. 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" [42]. 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. DOTS also shows poor compression ratios in these tests. This is related to LISSED, a cumulative error measure that DOTS uses, in which each point contributes to the error, such that the LISSED error bound of ϵ^2 may limit this algorithm to compress more points into a line segment, while those points may be compressed *w.r.t.* the SED error bound of ϵ.

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 are 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.

Moreover, weak algorithms OPERB-A and CISED-W typically have a few advantages in terms of compression ratios compared with their strong simplification counterparts OPERB and CISED for the cases with relatively small error bounds (*e.g.,* $\epsilon < 40$ meters in these tests).

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. This is because SED saves temporal information while PED does not. More specifically, *the output sizes of using* SED *are approximately twice of* PED. As shown in Figures 16 and 17, 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. Although the loss of temporal information may lead to unexpected results, *e.g.,* unbounded answer-errors to spatio-temporal queries (see Section 6.3.5), this price is worthwhile for certain applications.

Secondly, we find that SED has obviously better compression ratios than DAD on 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. Indeed, it is hard to compare SED and DAD under absolutely fair conditions, as SED is a Euclidean distance metric, having a value in $[0, \infty]$ meters, and DAD is a direction metric, having a value in $[0, 360)$ degrees. Hence, we choose to consider more practical scenarios, *i.e.,* one uses SED with $\epsilon \leq 100$ meters, and the other uses DAD with $\epsilon \leq 60$ degrees, and we compare the performance of SED with $\epsilon = 100/k$ meters *vs.* DAD with $\epsilon = 60/k$ degrees with $k \geq 1$. We also follow this when comparing SED and DAD in the sequel.

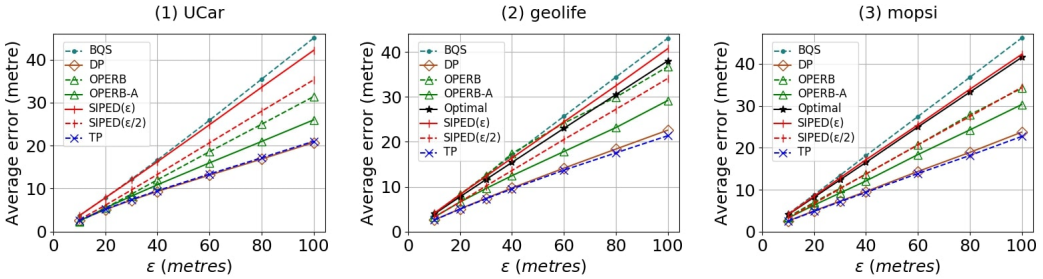

Fig. 22. Evaluation of average errors (PED) on full datasets: varying the error bound ϵ.

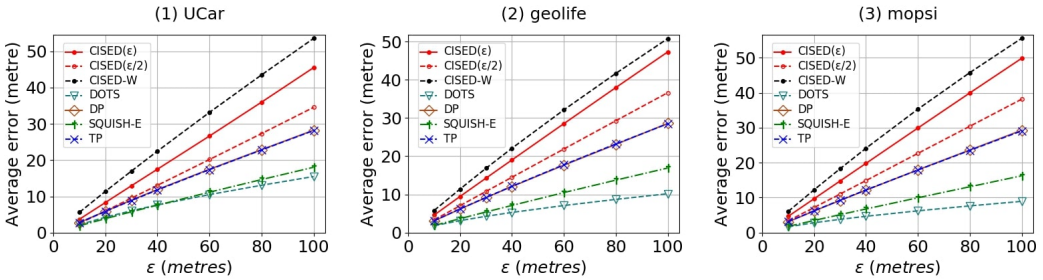

Fig. 23. Evaluation of average errors (SED) on full datasets: varying the error bound ϵ.

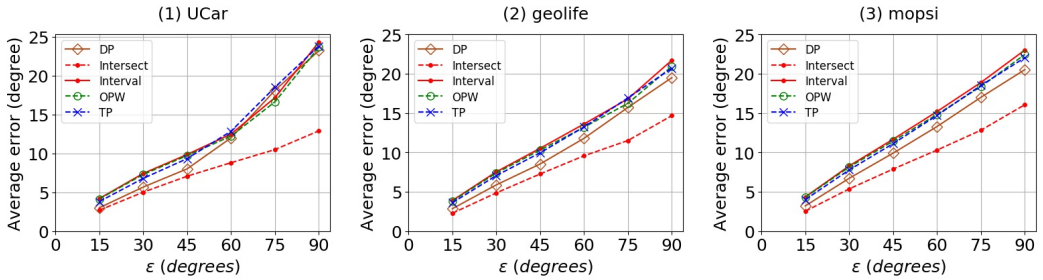

Fig. 24. Evaluation of average errors (DAD) on full datasets: varying the error bound ϵ.

6.3.2 Evaluation and Analysis of Average Simplification Error. The average simplification errors of these algorithms, under varied error bounds ϵ and trajectory sizes, are reported in Figures 22, 23, 24, 25, 26 and 27. We first report our findings.

(1) Datasets and data sizes 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 algorithm OPERB-A, 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 OPERB-A, SIPED (ϵ) and BQS are on average (71.17%, 80.10%, 69.82%), (100.05%, 101.01%, 102.69%) and (104.67%, 108.91%, 106.92%) of Optimal on datasets (UCar, Geolife, Mopsi), respectively. For example, the average errors of algorithms (Optimal, TP, DP, BQS, SIPED (ϵ), SIPED ($\frac{\epsilon}{2}$), OPERB, OPERB-A) in the full Mopsi are (16.08, 9.19, 9.68, 17.4, 12.96, 16.83, 12.77, 12.45) metres when $\epsilon = 40m$.

(3) When using SED, the average errors from the smallest to the largest are online algorithm

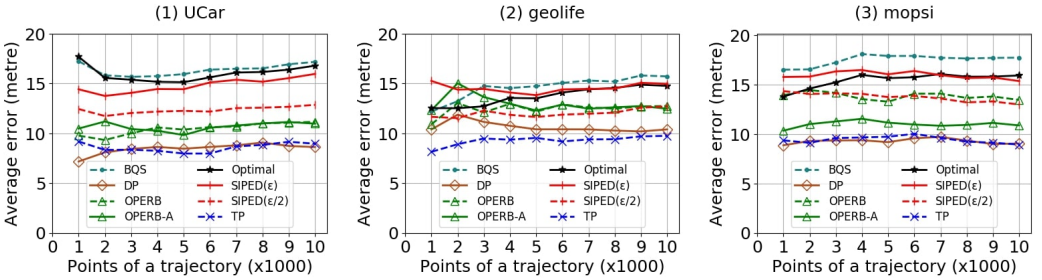

Fig. 25. Evaluation of average errors (PED) on small datasets: varying the size of trajectories.

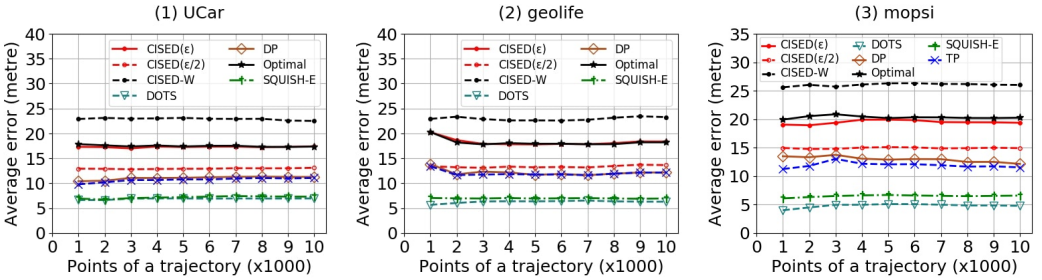

Fig. 26. Evaluation of average errors (SED) on small datasets: varying the size of trajectories.

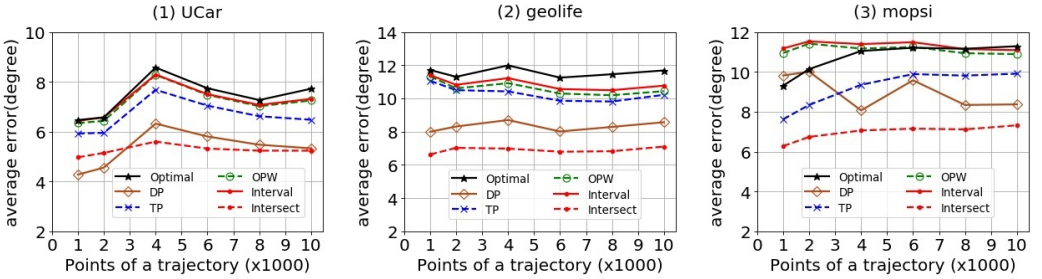

Fig. 27. Evaluation of average errors (DAD) on small datasets: varying the size of trajectories.

DOTS, online algorithm SQUISH-E, batch algorithms TP and DP, one-pass algorithm CISED ($\frac{\epsilon}{2}$), the Optimal algorithm and one-pass algorithm CISED (ϵ), and one-pass algorithm CISED-W. 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-W, CISED (ϵ), CISED ($\frac{\epsilon}{2}$), SQUISH-E and DOTS are on average (115.24%, 120.34%, 123.40%), (97.32%, 106.74%, 108.16%), (75.29%, 76.03%, 81.44%), (40.61%, 38.15%, 34.22%) and (39.12%, 28.80%, 23.90%) of Optimal on datasets (UCar, Geolife, Mopsi), respectively. For example, the average errors of algorithms (Optimal, TP, DP, SQUISH-E, DOTS, CISED (ϵ), CISED ($\frac{\epsilon}{2}$), CISED-W) in full Mopsi are (19.39, 12.17, 12.20, 6.76, 4.67, 20.68, 14.71, 24.12) 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 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 DOTS and SQUISH-E have 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, considerably 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 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. 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 Analysis of Efficiency. In this set of tests, we compare the efficiency of these algorithms. The results are reported in Figures 28, 29, 30, 31, 32 and 33. 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 except DOTS.

(2) When using PED, in most cases, the running time from the smallest to the largest is one-pass algorithms SIPED, OPERB and OPERB-A, batch algorithms TP and DP, and online algorithm BQS. Algorithms SIPED ($\frac{\epsilon}{2}$), OPERB and OPERB-A 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, Mopsi), respectively. For example, in Mopsi, the running time of algorithms (TP, DP, BQS, SIPED (ϵ), SIPED ($\frac{\epsilon}{2}$), OPERB, OPERB-A) is (232.9, 124.2, 469.4, 6.89, 7.6, 8.6, 9.4) seconds when $\epsilon = 40m$.

(3) When using SED, the running time of algorithms except DOTS from the smallest to the largest is one-pass algorithms CISED and CISED-W, online algorithm SQUISH-E, and batch algorithms TP and DP, while DOTS has varied running time. In full datasets, DOTS runs faster than, comparable with and slower than DP in datasets UCar, Geolife and Mopsi, respectively, while in small datasets, it often runs faster than DP. Algorithm CISED (ϵ) is (1.17, 1.17, 0.91) times of CISED ($\frac{\epsilon}{2}$), 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 ($\frac{\epsilon}{2}$) on datasets (UCar, Geolife, Mopsi), respectively. For example, in Mopsi, the running time of algorithms (TP, DP, DOTS, SQUISH-E, CISED (ϵ), CISED ($\frac{\epsilon}{2}$), CISED-W) is (156.6, 104.8, 361.1, 27.2, 11.6, 9.7, 9.7) seconds when $\epsilon = 40m$.

(4) When using DAD, one-pass algorithms Intersect and Interval run much faster than batch algorithms TP and DP and online algorithm OPW. Algorithm Interval is (1.80, 1.84, 1.81) times

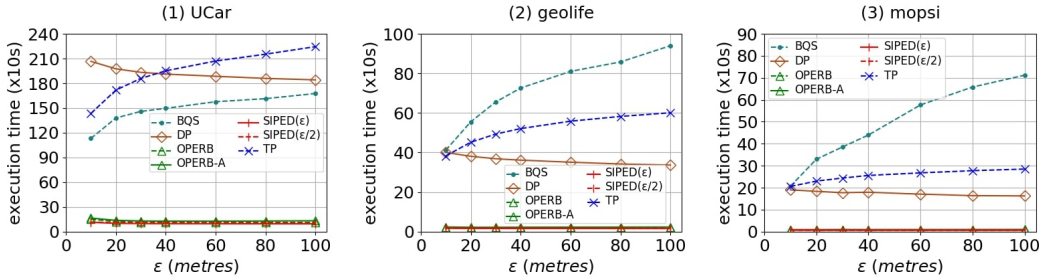

Fig. 28. Evaluation of running time (PED) on full datasets: varying the error bound ϵ.

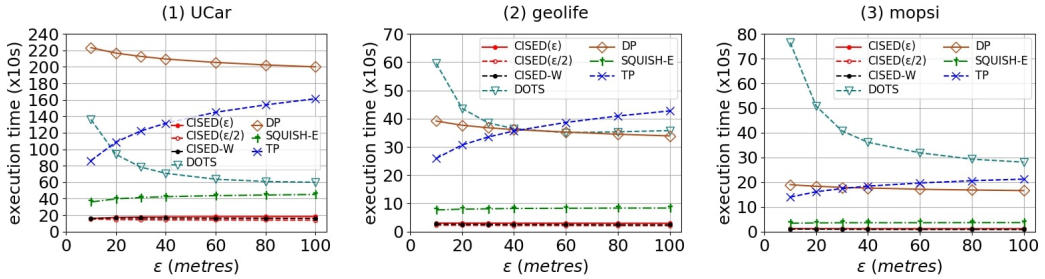

Fig. 29. Evaluation of running time (SED) on full datasets: varying the error bound ϵ.

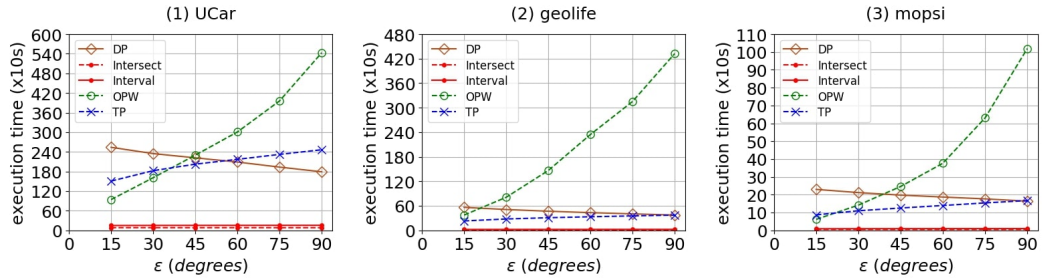

Fig. 30. Evaluation of running time (DAD) on full datasets: varying the error bound ϵ.

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

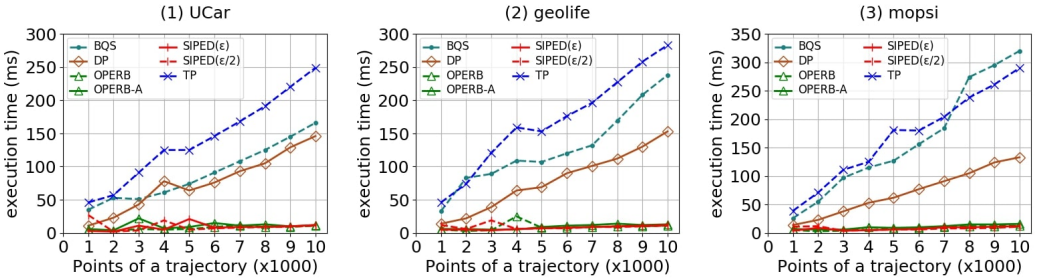

Fig. 31. Evaluation of running time (PED) on small datasets: varying the size of trajectories.

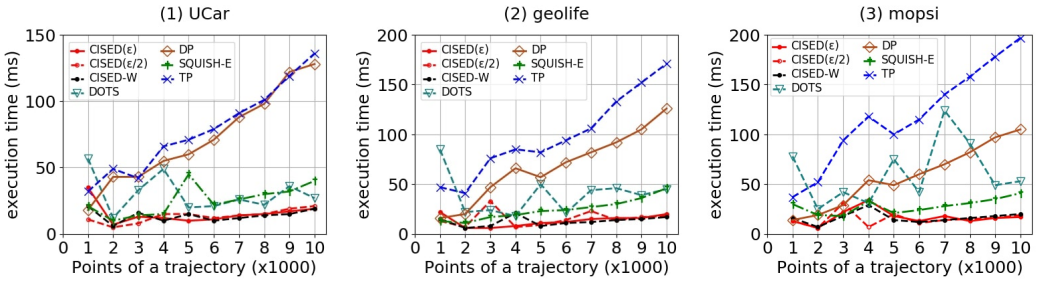

Fig. 32. Evaluation of running time (SED) on small datasets: varying the size of trajectories.

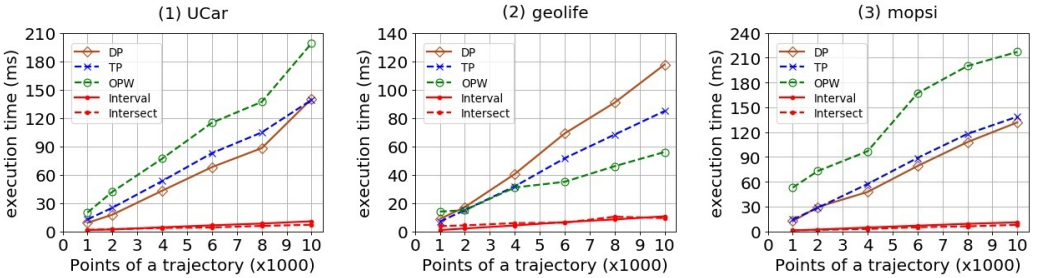

Fig. 33. Evaluation of running time (DAD) on small datasets: varying the size of trajectories.

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. DOTS runs very slow on the full datasets, partially because its frequent copying of memory wastes a lot of time and becomes its bottleneck of efficiency.

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.

As we analyzed above, different algorithms show different trends with error bounds because they adopt different routines and principles for trajectory compression. Further, weak simplification

Table 5. The max errors of algorithms in data aging that set $\epsilon_1 = 40m$ (or $\epsilon_1 = 30^o$ when using DAD) in the first run and $\epsilon_2 = 60m$ (or $\epsilon_2 = 50^o$ 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	DOTS	98.47	95.92	96.7	/	-	-	-
OPERB-A	99.99	99.99	99.99	CISED-W	99.21	99.34	98.81	/	-	-	-

Table 6. The final compression ratios of algorithms in data aging that set $\epsilon_1 = 40m$ (or $\epsilon_1 = 30^o$ when using DAD) in the first run and $\epsilon_2 = 60m$ (or $\epsilon_2 = 50^o$ 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	DOTS	11.81	3.89	2.55	/	-	-	-
OPERB-A	5.53	1.90	1.93	CISED-W	8.94	3.76	2.34	/	-	-	-

Table 7. The compression ratios of algorithms running on the raw trajectories that set $\epsilon_3 = \epsilon_1 + \epsilon_2 = 100m$ (or $\epsilon_3 = 80^o$ 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	2.69	1.91	DP	8.25	15.17	17.04
TP	3.70	1.22	1.50	TP	7.56	2.70	1.73	TP	2.47	7.51	9.42
BQS	3.23	1.71	1.39	SQUISH-E	10.36	4.10	2.82	OPW	3.23	8.66	10.75
SIPED (ϵ)	3.25	1.04	1.28	CISED (ϵ)	6.68	2.35	1.49	Interval	2.79	8.01	9.95
SIPED ($\frac{\epsilon}{2}$)	3.81	1.29	1.55	CISED ($\frac{\epsilon}{2}$)	8.02	2.90	1.87	Intersect	4.32	11.75	13.61
OPERB	4.08	1.43	1.56	DOTS	10.91	4.33	3.43	/	-	-	-
OPERB-A	4.06	1.31	1.20	CISED-W	6.92	2.98	1.77	/	-	-	-

algorithms have similar running time to their corresponding strong simplification algorithms as they share the same key routines for trajectory compression.

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 Analysis of Data Aging. In this set of tests, we compare the errors and compression ratios of algorithms in data aging. We set $\epsilon_1 = 40m$ (or $\epsilon_1 = 30^o$) in the first run and $\epsilon_2 = 60m$ (or $\epsilon_2 = 50^o$) in the second run. Besides, we also run these algorithms on the raw trajectories, setting $\epsilon_3 = \epsilon_1 + \epsilon_2 = 100m$ (or $\epsilon_3 = 80^o$). 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 $\epsilon_2 = 50^o$, confirming that they are not aging friendly and their max errors are consistent with Theorems 5.2, 5.3 and 5.4.

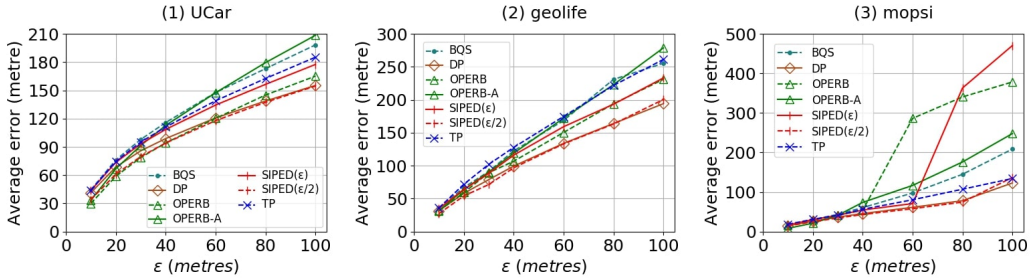

Fig. 34. Evaluation of *where_at* queries (PED) on full datasets: varying error bound ϵ.

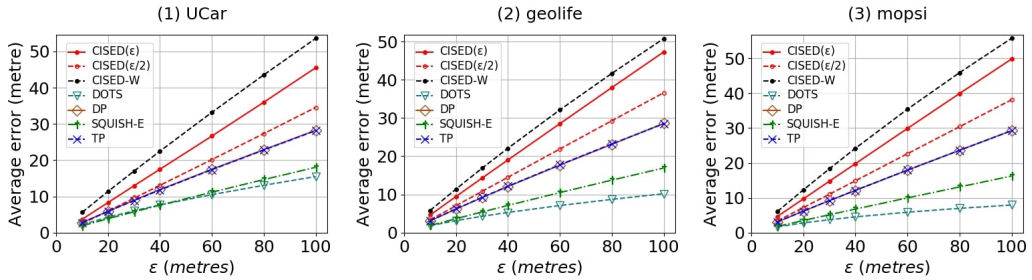

Fig. 35. Evaluation of *where_at* queries (SED) on full datasets: varying error bound ϵ.

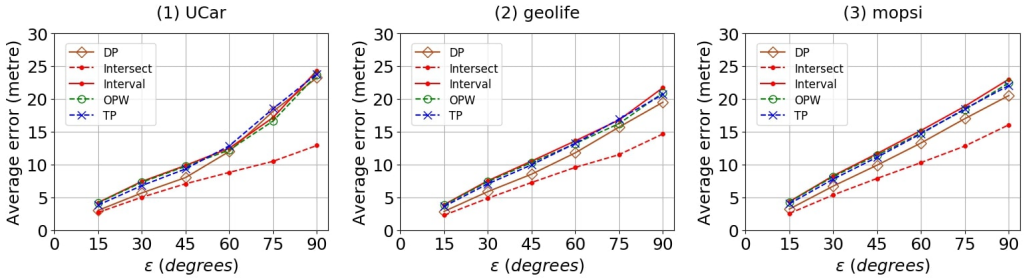

Fig. 36. Evaluation of *where_at* queries (DAD) on full datasets: varying error bound ϵ.

(2) Algorithm DP using DAD and other algorithms have max error less than $\epsilon_1 + \epsilon_2 = 100m$ or $\epsilon_1 + \epsilon_2 = 80^o$, 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 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, and all algorithms have bounded errors. Indeed, aging friendliness 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.

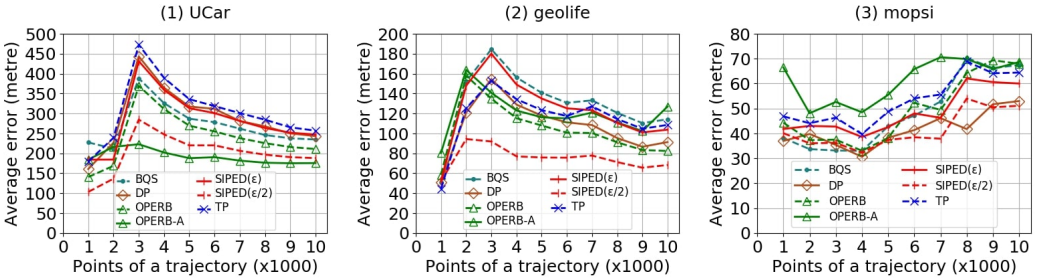

Fig. 37. Evaluation of *where_at* queries (PED) on small datasets: varying the size of trajectories.

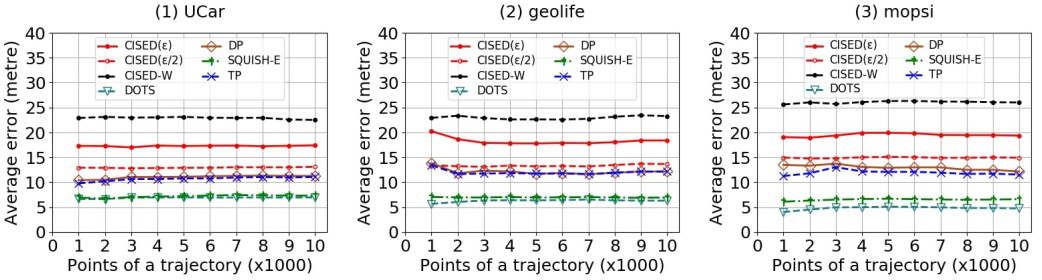

Fig. 38. Evaluation of *where_at* queries (SED) on small datasets: varying the size of trajectories.

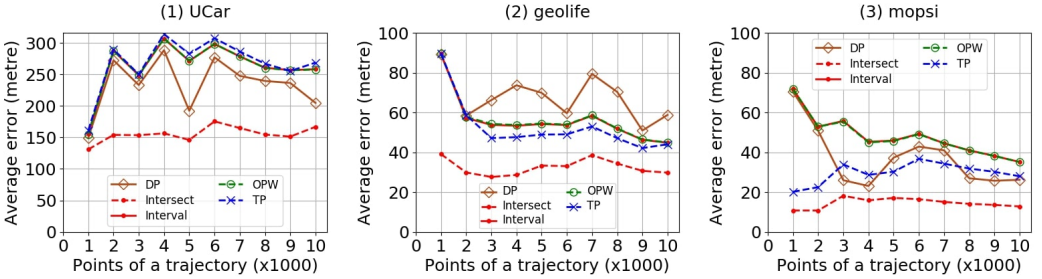

Fig. 39. Evaluation of *where_at* queries (DAD) on small datasets: varying the size of trajectories.

6.3.5 Evaluation and Analysis 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* [3, 60]. Among them, *where_at* query, *i.e.,* "*the position P of a moving object at time t*" [3], is the foundation of *range* and *nearest_neighbor* queries, and *when_at* query, *i.e.,* "*the time t at which a moving object on a trajectory is expected to be at position P*" [3], is also a critical building block for many applications. Hence, we choose them to evaluate compressed trajectories simplified by LS algorithms using PED, SED and DAD. As mentioned in [3, 60], 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.

Exp-1: *where_at* queries. We first compress these trajectories using PED, SED and DAD, respectively. Then, for each point P in an original trajectory $\overddot{T}$, we perform a *where_at* 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' to denote the error of queries. The max and average errors of the queries are reported in Table 8 and Figures 34, 35, 36, 37,38 and 39, respectively.

Table 8. The max errors of *where_at* queries on the compressed trajectories: fixed $\epsilon = 40m$ or 30^o when using DAD.

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	/	-	-	-
OPERB-A	7.50×10^6	1.84×10^6	5.03×10^6	/	-	-	-

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

Table 9. The max errors ($\times 10^6$ s) of *when_at* queries on compressed trajectories: fixed $\epsilon = 40m$ or 30^o when using DAD.

Alg. (PED)	UCar	Geolife	Mopsi	**Alg. (SED)**	UCar	Geolife	Mopsi	**Alg. (DAD)**	UCar	Geolife	Mopsi
DP	11.2	126	6.59	DP	11.6	5.01	1.84	DP	17.6	5.88	3.99
TP	26.1	126	9.91	TP	11.6	5.01	5.18	TP	17.6	2.90	5.06
BQS	11.6	126	11.5	SQUISH-E	11.6	18.3	2.62	OPW	8.19	4.02	5.06
SIPED (ϵ)	11.5	73.7	6.65	CISED (ϵ)	11.6	39.7	5.48	Interval	8.19	4.02	5.06
SIPED ($\frac{\epsilon}{2}$)	11.5	73.7	6.68	CISED ($\frac{\epsilon}{2}$)	11.6	39.7	5.20	Intersect	8.19	2.63	5.06
OPERB	21.6	75.2	8.31	DOTS	11.6	18.3	2.91	/	-	-	-
OPERB-A	22.1	126	11.5	CISED-W	11.6	47.7	5.50	/	-	-	-

(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.
(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 few points having query errors larger than error bounds, thus, it has the smallest average query errors in all algorithms and all datasets.

Exp-2: *when_at* queries. For each point P in an original trajectory $\ddot{\mathcal{T}}$, we do a *when_at* query on the simplified trajectory. Since quite a few original data points are not exactly on the line segments of the simplified trajectory, we first map each original point P to its closest point P' having the minimum Euclidean distance on the line segment representing this point along the same way as [3], then we get the expected time $P'.t'$ of P' *w.r.t.* this line segment in a way inverse to the finding of a synchronized point, and we finally calculate the absolute time difference between the actual time $P.t$ and the expected time $P'.t'$ as the error of the query. The max and average errors of the queries are reported in Table 9 and Figures 40, 41, 42, 43, 44 and 45, respectively.
(1) The max error of *when_at* queries on simplified trajectories is unbounded as proved in [3]. Indeed, the query error is related to the length of the time interval between two neighboring points of a simplified trajectory. We explain this with an example. Suppose that one moves from the front door, quickly passes through the living room, and goes to bed for a long sleep. Since the house is not big, the trajectory is possible simplified to two points, *i.e.,* P_s in the front door and P_e on the bed. In this case, a *when_at* query taking a point P in his living room as input may return the excepted time t' with an error of several hours to the actual time $P.t$.
(2) The average error of *when_at* queries normally increases with the increment of error bounds. This is obvious as the increment of error bounds enlarges the average length of the time interval between two neighboring points in a simplified trajectory, which leads to the increase of the average

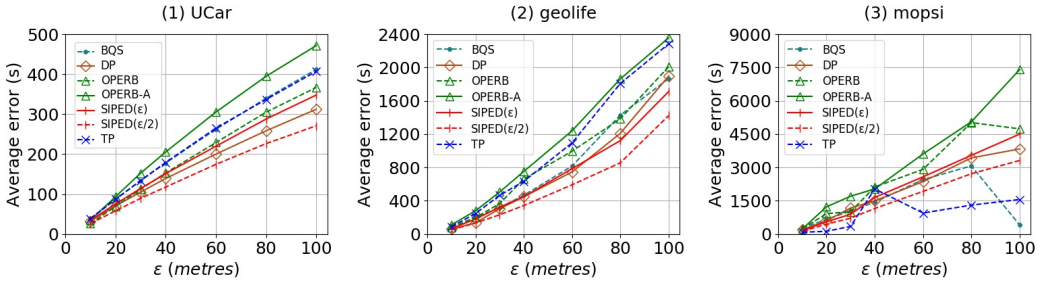

Fig. 40. Evaluation of *when_at* queries (PED) on full datasets: varying error bound ϵ.

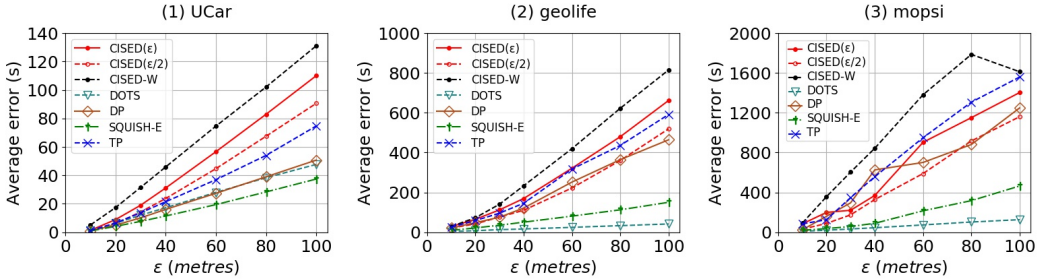

Fig. 41. Evaluation of *when_at* queries (SED) on full datasets: varying error bound ϵ.

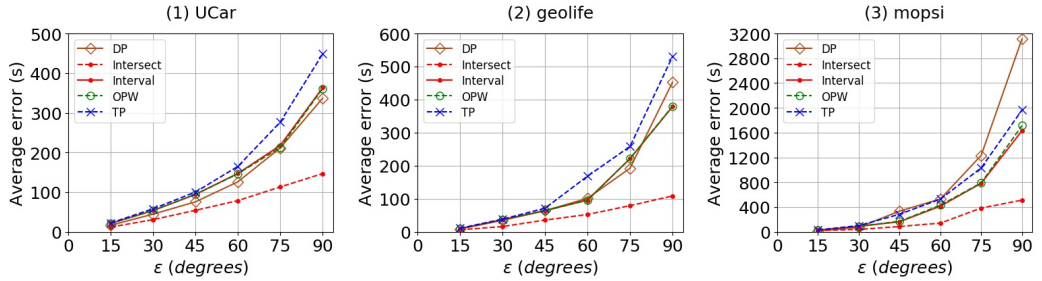

Fig. 42. Evaluation of *when_at* queries (DAD) on full datasets: varying error bound ϵ.

error of *when_at* queries. Similarly, the algorithms having better compression ratios typically have larger errors for *when_at* queries.

(3) SED algorithms have smaller average *when_at* query errors than PED algorithms, majorly because SED introduces smaller distance errors and has relatively worse compression ratios compared with PED. DAD algorithms also have smaller average *when_at* query errors than PED algorithms, partially because of their relatively worse compression ratios compared with PED.

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

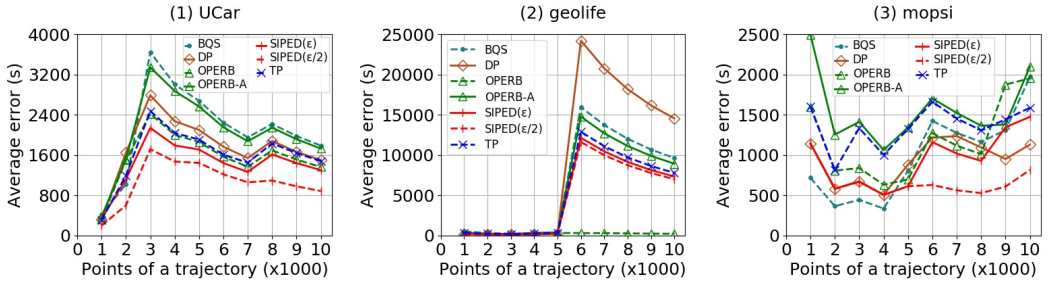

Fig. 43. Evaluation of *when_at* queries (PED) on small datasets: varying the size of trajectories.

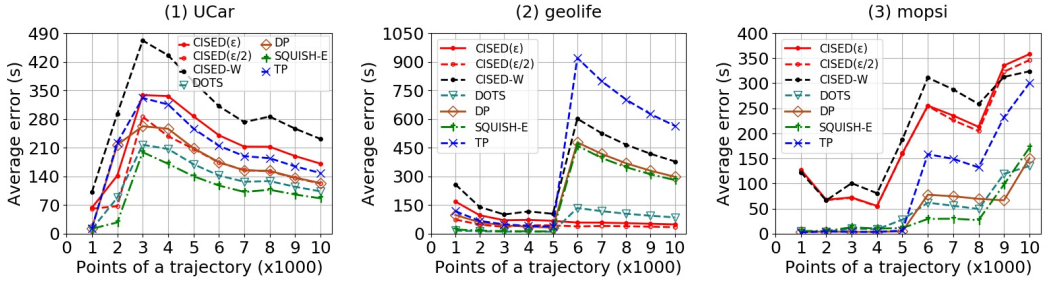

Fig. 44. Evaluation of *when_at* queries (SED) on small datasets: varying the size of trajectories.

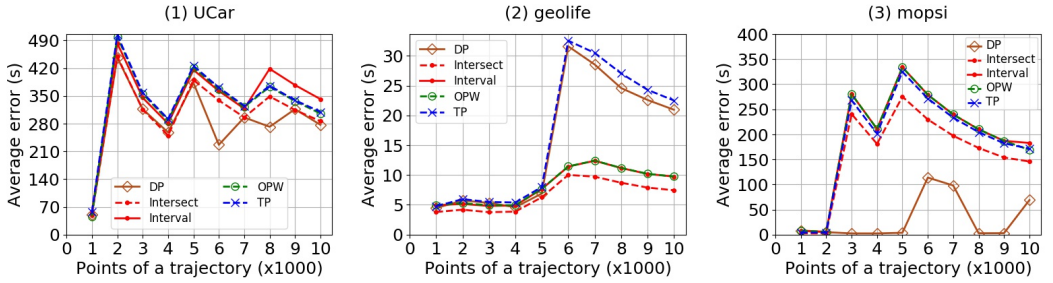

Fig. 45. Evaluation of *when_at* queries (DAD) on small datasets: varying the size of trajectories.

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, and algorithm DOTS has poor compression ratios, the best average errors and varied efficiency. In our tests (in Java), when DOTS is processing a long trajectory, the frequent copying of memory makes it even slower than batch algorithms. *(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 has better compression ratios and also larger average errors than the half ϵ *sector/cone/range*. Weak simplifications show a bit better compression ratios compared with strong simplifications when the SED/PED error bounds are relatively small, *e.g.,* less than 40 meters in the tests. 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)* 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 $\epsilon_1 + \epsilon_2$. Moreover, algorithms DP using PED and SED are *aging friendly*, while others are not.

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 queries friendly *w.r.t. where_at* queries, while the others are not. Note that, *though algorithms using* PED *and* DAD *are error bounded by* PED *and* DAD*, respectively, they are not able to guarantee the error bounds of spatio-temporal queries.* Actually, they might lead to very large query errors. That is, all these algorithms and distance metrics become unbound for *when_at* queries.

Comparing with the recent experimental study [66], this work has the following new findings: (1) the compression ratios and average errors of optimal, batch, online and one-pass algorithms *w.r.t.* distance metrics (PED, SED and DAD), error bounds and data sizes are systematically studied, (2) the efficiency of algorithms *w.r.t.* distance metrics, error bounds and data sizes are studied by implementing all algorithms in the same programming language. Our tests reveal that the running times from the fastest to the slowest are typically Intersect, Interval, OPERB, SQUISH-E, DP, DOTS and BQS, which is partially different from [66] that are Intersect, Interval, DP, OPERB, DOTS, SQUISH-E and BQS, (3) aging friendliness and errors are investigated, which is indeed new from all previous experimental studies, (4) one-pass algorithm SIPED (ϵ) is efficient and has good compression ratios, and one-pass algorithm CISED, either CISED (ϵ) or CISED-W, is better than batch and online algorithms in terms of compression ratios and efficiency.

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) and CISED-W using SED, and TP and Interval using DAD are approximately 103%–124%, 101%–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) and CISED-W using SED, and TP and Interval using DAD are good candidates. In case average errors are concerned, then online algorithm DOTS and 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 better or comparable compression ratios compared with*

batch and online algorithms. Hence, they are the prominent trajectory compression algorithms when average errors are not the main concern.

Besides, 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 are 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 *w.r.t.* the *where_at*, *range* and *nearest_neighbor* queries. Hence, it is the best choice for such applications, and all of them are not friendly *w.r.t.* the *when_at* query. Further, the choice of a distance metric has impacts on the performance. For compression ratios, the use 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 obviously 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: SEMANTIC BASED TRAJECTORY COMPRESSION ALGORITHMS

Apart from the techniques evaluated in this article, there exist other approaches for various requirements of trajectory compression. We introduce the semantic based methods, one of the most important types of methods.

The trajectories of certain moving objects in urban areas, such as cars and trucks, are constrained by road networks. Hence, there are trajectory compression methods based on road networks [2, 9, 10, 18, 21, 24, 28, 47, 56] that first project trajectory points onto roads (known as map-matching [49]), then compress the matched data points by *piece-wise linear approximation* methods [14, 35, 45, 65]. Note that (1) map-matching is thought an effective way to improve the quality of a raw uncertain trajectory [28], and (2) dilution-matching-encoding [17] is an exception of such methods in that it first simplifies the original trajectory points by some line simplification method, then it projects the simplified points onto roads. Some methods [52, 53] compress trajectories making use of other domain knowledge, such as places of interests (POI) along the trajectories [52], and the high frequency patterns of compressed trajectories instead of roads [17, 18, 26, 56], to improve the compression effectiveness. The location information from sensors is usually imprecise, and may be noisy and error prone [2], while map-matching is believed able to correct the error by "snapping" data points onto the road network. Further, they are able to represent a trajectory basing on long paths (a path is a sequence of roads connected one by one), and to further mine the frequency patterns, so as to improve the overall compression of trajectories. However, users should keep in mind that the effectiveness of these semantic based methods highly depends on the quality of semantic information (*e.g.,* road networks) and sequence labeling (*e.g.,* map-matching) algorithms. In some cases, *e.g.,* when the road network is not up to date or there is no road nearby or moving objects actually move along the parallel roads (but outside of the roads), these methods may result in incorrect map-matching, and introduce extra errors.

We believe that the semantic based methods and the line simplification methods are orthogonal, and may be combined with each other to improve the effectiveness of trajectory compression [29], like the way that dilution-matching-encoding [17] does.

ACKNOWLEDGMENTS

This work is supported in part by National Key Research and Development Program (2016YFB1000103), NSFC (61925203) and SKLSDE (2020ZX-31). For any correspondence, please refer to Shuai Ma.

REFERENCES

[1] Gill Barequet, Danny Z. Chen, Ovidiu Daescu, Michael T. Goodrich, and Jack Snoeyink. 2002. Efficiently approximating polygonal paths in three and higher dimensions. *Algorithmica* 33, 2 (2002), 150–167.
[2] Hu Cao and Ouri Wolfson. 2005. Nonmaterialized Motion Information in Transport Networks. In *ICDT*.
[3] Hu Cao, Ouri Wolfson, and Goce Trajcevski. 2006. Spatio-temporal data reduction with deterministic error bounds. *VLDBJ* 15, 3 (2006), 211–228.
[4] W. Cao and Y. Li. 2017. DOTS: An online and near-optimal trajectory simplification algorithm. *Journal of Systems and Software* 126(Supplement C) (2017), 34–44.
[5] W. Chan and F. Chin. 1996. Approximation of polygonal curves with minimum number of line segments. *IJCGA* 6, 1 (1996), 378–387.
[6] Danny Z. Chen and Ovidiu Daescu. 2002. Space-efficient algorithms for approximating polygonal curves in two dimensional space. *IJCGA* 13, 2 (2002), 95–111.
[7] Minjie Chen, Mantao Xu, and Pasi Fränti. 2012. Compression of GPS trajectories. In *DCC*.
[8] Minjie Chen, Mantao Xu, and Pasi Fränti. 2012. A Fast Multiresolution Polygonal Approximation Algorithm for GPS Trajectory Simplification. *TIP* 21, 5 (2012), 2770–2785.
[9] Yukun Chen, Kai Jiang, Yu Zheng, Chunping Li, and Nenghai Yu. 2009. Trajectory simplification method for location-based social networking services. In *LBSN*.
[10] Alminas Civilis, Christian S. Jensen, and Stardas Pakalnis. 2005. Techniques for efficient road-network-based tracking of moving objects. *TKDE* 17, 5 (2005), 698–712.
[11] Ovidiu Daescu and Ningfang Mi. 2005. Polygonal chain approximation: a query based approach. *Computational Geometry* 30 (2005), 41–58.
[12] David H Douglas and Thomas K Peucker. 1973. 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.
[13] James George Dunham. 1986. Optimum Uniform Piecewise Linear Approximation of Planar Curves. *TPAMI* 8, 1 (1986), 67–75.
[14] Hazem Elmeleegy, Ahmed K. Elmagarmid, Emmanuel Cecchet, Walid G. Aref, and Willy Zwaenepoel. 2009. Online Piece-wise Linear Approximation of Numerical Streams with Precision Guarantees. *PVLDB* 2, 1 (2009), 145–156.
[15] D. Eu and G. T. Toussaint. 1994. On approximation polygonal curves in two and three dimensions. *CVGIP: Graphical Models and Image Processing* 56, 3 (1994), 231–246.
[16] Oliviu Ghica, Goce Trajcevski, Ouri Wolfson, Ugo Buy, Peter Scheuermann, Fan Zhou, and Dennis Vaccaro. 2010. Trajectory Data Reduction in Wireless Sensor Networks. *IJNGC* 1, 1 (2010), 28–51.
[17] Ranit Gotsman and Yaron Kanza. 2015. A Dilution-matching-encoding compaction of trajectories over road networks. *GeoInformatica* 19, 2 (2015), 331–364.
[18] Yunheng Han, Weiwei Sun, and Baihua Zheng. 2017. COMPRESS: A Comprehensive Framework of Trajectory Compression in Road Networks. *TODS* 42, 2 (2017), 11:1–11:49.
[19] Henrik Hargitai, Jue Wang, Philip J. Stooke, Irina Karachevtseva, Akos Kereszturi, and Mátyás Gede. 2017. Map Projections in Planetary Cartography. *Lecture Notes in Geoinformation and Cartography, Springer International Publishing* (2017).
[20] John Hershberger and Jack Snoeyink. 1992. Speeding up the Douglas-Peucker line-simplification algorithm. *Technical Report, University of British Columbia* (1992).
[21] Chih Chieh Hung, WenChih Peng, and WangChien Lee. 2015. Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. *VLDBJ* 24, 2 (2015), 169–192.
[22] Hiroshi Imai and Masao Iri. 1986. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics, and Image Processing* 36 (1986), 31–41.
[23] Bingqing Ke, Jie Shao, and Dongxiang Zhang. 2017. An Efficient Online Approach for Direction-Preserving Trajectory Simplification with Interval Bounds. In *MDM*.
[24] Georgios Kellaris, Nikos Pelekis, and Yannis Theodoridis. 2013. Map-matched trajectory compression. *The Journal of Systems and Software* 86 (2013), 1566–1579.
[25] Eamonn J. Keogh, Selina Chu, David M. Hart, and Michael J. Pazzani. 2001. An Online Algorithm for Segmenting Time Series. In *ICDM*.

[26] Satoshi Koide, Yukihiro Tadokoro, Chuan Xiao, and Yoshiharu Ishikawa. 2018. CiNCT: Compression and Retrieval for Massive Vehicular Trajectories via Relative Movement Labeling. In *ICDE*.
[27] Ralph Lange, Frank Dürr, and Kurt Rothermel. 2011. Efficient real-time trajectory tracking. *VLDBJ* 20, 5 (2011), 671–694.
[28] Tianyi Li, Ruikai Huang, Lu Chen, Christian S. Jensen, and Torben Bach Pedersen. 2020. Compression of Uncertain Trajectories in Road Networks. In *PVLDB*.
[29] Xuelian Lin, Jiahao Jiang, Shuai Ma, Yimeng Zuo, and Chunming Hu. 2019. One-Pass Trajectory Simplification Using the Synchronous Euclidean Distance. *VLDBJ* 28, 6 (2019), 897–921.
[30] Xuelian Lin, Shuai Ma, Han Zhang, Tianyu Wo, and Jinpeng Huai. 2017. One-Pass Error Bounded Trajectory Simplification. *PVLDB* 10, 7 (2017), 841–852.
[31] Jiajun Liu, Kun Zhao, Philipp Sommer, Shuo Shang, Brano Kusy, and Raja Jurdak. 2015. Bounded Quadrant System: Error-bounded trajectory compression on the go. In *ICDE*.
[32] Jiajun Liu, Kun Zhao, Philipp Sommer, Shuo Shang, Brano Kusy, Jae-Gil Lee, and Raja Jurdak. 2016. A Novel Framework for Online Amnesic Trajectory Compression in Resource-constrained Environments. *TKDE* 28, 11 (2016), 2827–2841.
[33] Cheng Long, Raymond Chi-Wing Wong, and H.V. Jagadish. 2013. Direction-preserving trajectory simplification. *PVLDB* 6, 10 (2013), 949–960.
[34] Cheng Long, Raymond Chi-Wing Wong, and H.V. Jagadish. 2014. Trajectory Simplification: On Minimizing the Direction-based Error. *PVLDB* 8, 1 (2014), 49–60.
[35] Ge Luo, Ke Yi, Siu-Wing Cheng, Zhenguo Li, Wei Fan, Cheng He, and Yadong Mu. 2015. Piecewise Linear Approximation of Streaming Time Series Data with Max-error Guarantees. In *ICDE*.
[36] R. Mariescu-Istodor and P. Fränti. 2017. Grid-based method for GPS route analysis for retrieval. *ACM Transactions on Spatial Algorithms and Systems* 3, 3 (2017).
[37] Jean Damascene Mazimpaka and Sabine Timpf. 2016. Trajectory data mining: A review of methods and applications. *Journal of Spatial Information Science* 13 (2016), 61–99.
[38] Avraham Melkman and Joseph O'Rourke. 1988. On Polygonal Chain Approximation. *Machine Intelligence and Pattern Recognition* 6 (1988), 87–95.
[39] Nirvana Meratnia and Rolf A. de By. 2004. Spatiotemporal Compression Techniques for Moving Point Objects. In *EDBT*.
[40] Rohit Metha and V.K.Mehta. 1999. *The Principles of Physics.* S Chand.
[41] Jonathan Muckell, Jeong-Hyon Hwang, Catherine T. Lawson, and S. S. Ravi. 2010. Algorithms for compressing GPS trajectory data: an empirical evaluation. In *ACM-GIS*.
[42] Jonathan Muckell, Jeong-Hyon Hwang, Vikram Patil, Catherine T. Lawson, Fan Ping, and S. S. Ravi. 2011. SQUISH: an online approach for GPS trajectory compression. In *COM.Geo*.
[43] Jonathan Muckell, Paul W. Olsen, Jeong-Hyon Hwang, Catherine T. Lawson, and S. S. Ravi. 2014. Compression of trajectory data: a comprehensive evaluation and new approach. *GeoInformatica* 18, 3 (2014), 435–460.
[44] Aiden Nibali and Zhen He. 2015. Trajic: An Effective Compression System for Trajectory Data. *TKDE* 27, 11 (2015), 3138–3151.
[45] Joseph O'Rourke. 1981. An On-Line Algorithm for Fitting Straight Lines Between Data Ranges. *Commun. ACM* 24, 9 (1981), 574–578.
[46] Theodosios Pavlidis and Steven L. Horowitz. 1974. Segmentation of Plane Curves. *TOC* 23, 8 (1974), 860–870.
[47] Iulian Sandu Popa, Karine Zeitouni, VincentOria, and Ahmed Kharrat. 2014. Spatio-temporal compression of trajectories in road networks. *GeoInformatica* 19, 1 (2014), 117–145.
[48] Michalis Potamias, Kostas Patroumpas, and Timos K. Sellis. 2006. Sampling Trajectory Streams with Spatiotemporal Criteria. In *SSDBM*.
[49] M.A. Quddus, W.Y. Ochieng, and R.B. Noland. 2007. Current map-matching algorithms for transport applications: State-of-the art and future research directions. *Transportation research part c: Emerging technologies* 15, 5 (2007), 312–328.
[50] U. Ramer. 1972. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing* 1 (1972), 244–256.
[51] K. Reumann and A.P.M Witkam. 1974. Optimizing curve segmentation in computer graphics. In *International Computing Symposium*.
[52] Kai-Florian Richter, Falko Schmid, and Patrick Laube. 2012. Semantic trajectory compression: Representing urban movement in a nutshell. *JOSIS* 4, 1 (2012), 3–30.
[53] Falko Schmid, Kai-Florian Richter, and Patrick Laube. 2009. Semantic Trajectory Compression. In *SSTD*.
[54] Wenzhong Shi and Chuikwan Cheung. 2006. Performance Evaluation of Line Simplification Algorithms for Vector Generalization. *Cartographic Journal* 43, 1 (2006), 27–44.

[55] J. Sklansky and V. Gonzalez. 1980. Fast polygonal approximation of digitized curves. *Pattern Recognition* 12 (1980), 327–331.
[56] Renchu Song, Weiwei Sun, Baihua Zheng, and Yu Zheng. 2014. PRESS: A Novel Framework of Trajectory Compression in Road Networks. *PVLDB* 7, 9 (2014), 661–672.
[57] G. T. Toussaint. 1985. On the complexity of approximating polygonal curves in the plane. In *International Symposium on Robotics and Automation*.
[58] Goce Trajcevski. 2016. Compression of Spatio-temporal Data – Advanced Seminar. In *MDM*.
[59] Goce Trajcevski, Hu Cao, Peter Scheuermanny, Ouri Wolfsonz, and Dennis Vaccaro. 2006. On-line Data Reduction and the Quality of History in Moving Objects Databases. In *MobiDE*.
[60] G. Trajcevski, O.Wolfson, K. Hinrichs, and S. Chamberlain. 2004. Managing uncertainty in moving objects databases. *ACM Transactions on Database Systems (TODS)* 29, 3 (2004), 463–507.
[61] Robert Weibel. 1997. Generalization of spatial data: Principles and selected algorithms. *Algorithmic Foundations of Geographic Information Systems* (1997).
[62] Robert Weibel and Genevieve Dutton. 1999. Generalising spatial data and dealing with multiple representations. *Computer Science* (1999).
[63] Charles M. Williams. 1978. An efficient algorithm for the piecewise linear approximation of planar curves. *Computer Graphics and Image Processing* 8 (1978), 286–293.
[64] Charles M. Williams. 1981. Bounded Straight-Line Approximation of Digitized Planar Curves and Lines. *Computer Graphics and Image Processing* 16 (1981), 370–381.
[65] Qing Xie, Chaoyi Pang, Xiaofang Zhou, Xiangliang Zhang, and Ke Deng. 2014. Maximum error-bounded Piecewise Linear Representation for online stream approximation. *VLDBJ* (2014).
[66] Dongxiang Zhang, Mengting Ding, Dingyu Yang, Yi Liu, Ju Fan, and Heng Tao Shen. 2018. Trajectory Simplification: An Experimental Study and Quality Analysis. *PVLDB* 9, 11 (2018), 934–946.
[67] Liangbin Zhao and Guoyou Shi. 2019. A trajectory clustering method based on Douglas-Peucker compression and density for marine traffic pattern recognition. *Ocean Engineering* 172 (2019), 456–467.
[68] Zhiyuan Zhao and Alan Saalfeld. 1997. Linear-time sleeve-fitting polyline simplification algorithms. In *AutoCarto*.
[69] Yu Zheng, Xing Xie, and Wei-Ying Ma. 2010. GeoLife: A Collaborative Social Networking Service among User, Location and Trajectory. *IEEE Data Eng. Bull.* 33, 2 (2010), 32–39.