Line Simplification Algorithms for Trajectory Compression [Experiments and Analyses]

Xuelian Lin Shuai Ma* Jiahao Jiang Yanchen Hou Tianyu Wo Chunming Hu
Beijing Advanced Innovation Center for Big Data and Brain Computing, Beijing, China
SKLSDE Lab, Beihang University, Beijing, China
{linxl, mashuai, jiangjh, houyc, woty, hucm}@buaa.edu.cn

ABSTRACT

Nowadays, various sensors are collecting, storing and transmitting tremendous trajectory data, and it is known that raw trajectory data seriously wastes the storage, network bandwidth and computing resources. Line simplification algorithms are effective approaches to attacking this issue by compressing data points in a trajectory to a set of continuous line segments, and are commonly used in practice. This paper presents experiments and analyses of representative error bounded line simplification algorithms, including both 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 and running time) of error bounded LS algorithms with respect to trajectory sizes and error bounds. Our study reveals the characteristics of error bounded LS algorithm, which lead to guidelines for practitioners to decide which algorithms and distance metrics should be adopted for a specific application.

PVLDB Reference Format:
Xuelian Lin, Shuai Ma, Jiahao Jiang, Yanchen Hou, Tianyu Wo, and Chunming Hu. Line Simplification Algorithms for Trajectory Compression [Experiments and Analyses]. *PVLDB*, 12(xxx): xxxx-yyyy, 2019.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

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 is increasing rapidly. Sampling rates are also improved for acquiring more accurate position information, which leads to longer trajectories than before. Thus, transmitting and storing raw trajectory data consume a large amount of network, storage and computing resources [4,7,8,14,18,19,22,25,26,28,31,32], and trajectory compression techniques [4,7–9,11,14,18,19,22,25,26,31,33,35] have been developed to alleviate this situation.

Proceedings of the VLDB Endowment, Vol. 12, No. xxx
ISSN 2150-8097.
DOI: https://doi.org/10.14778/xxxxxxx.xxxxxxx

Due to the limitations (poor compression ratio and data reconstruction overhead) of lossless compression, lossy compression techniques have become the mainstream of trajectory compression [17,41]. Quite a few lossy trajectory compression techniques, most notably the piece-wise line simplification [4,7–9,11,14,18,25,33] solving the *min-#* problem [6,12,27], have been developed. The idea of piece-wise line simplification (LS) comes from computational geometry, whose target is to approximate a fine piece-wise linear curve with a coarse one (whose corresponding data points are a subset of the former), such that the maximum distance of the former to the latter is bounded by a user specified threshold. It is widely used due to its distinct advantages: (a) simple and easy to implement, (b) no need of extra knowledge and suitable for freely moving objects [28], and (c) bounded errors with good compression ratios.

Algorithm taxonomy. LS algorithms fall into two categories: *optimal* and *sub-optimal* algorithms. *Optimal* methods [6,12] are to find the minimum number of points or segments to represent the original polygonal lines *w.r.t.* an error bound ϵ, by transforms the problem to search for the shortest path of a graph built from the original trajectory. The optimal LS algorithms have relative high time/space complexities which make them impractical for large trajectory data. Hence, *sub-optimal* LS algorithms have been developed and/or introduced for trajectory compression, and they achieve better efficiency at an expense of outputting a little more data points. By the applied distance checking policies, sub-optimal algorithms can further be classified into batch, online and one-pass algorithms.

(1) *Batch algorithms* such as Douglas-Peucker [9,22] and Theo-Pavlidis [27] apply global distance checking policies such that all trajectory points need to be loaded before starting compression, and a point may be checked multiple times to compute its distance to the corresponding line segments.

(2) *Online algorithms* such as OPW [22], SQUISH-E [24] and BQS [18] apply local distance checking policies, and need not to have the entire trajectory ready before compressing. They restrict the checking within a window, but may still check a point multiple times during the process.

(3) *One-pass algorithms* such as OPERB [17], SIPED [10,34, 39,42], CISED [16], Intersect [19] and Interval [13] apply better local distance checking policies, which even do not need a window to buffer the previous 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 tech-

niques 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 as the distance metric, which brings good compression ratios [4, 8, 9, 11, 18, 25, 33]. (1) PED does not preserve the temporal information. Hence, SED was then introduced to preserve temporal information of trajectories and to support spatio-temporal queries [22]. (2) DAD [19, 41] was introduced to preserve the direction information of moving objects, and was initially called the *direction-based measurement* in [19]. It is important for applications such as trajectory clustering and direction-based query processing [19,20]. Note that LS algorithms using SED or DAD may produce more line segments than using PED. However, the use of SED and DAD further preserve temporal and direction information, respectively.

Motivations. Empirical studies on trajectory compression algorithms have been conducted [23, 25, 38]. However, they only discuss a small number of algorithms. The very recent study [41] does evaluate a wide range of trajectory simplification algorithms, and it provides a good experimental study on compression error and spatio-temporal query analyses. However, two important aspects for trajectory simplification (*i.e.,* compression ratios and running time) are not systematically studied. As a consequence, it remains difficult for practitioners to decide which algorithms and distance metrics should be adopted for a specific application.

Contributions. In this paper, we conduct a thorough and systematic evaluation and analyses of the mainstream trajectory compression techniques (*i.e.,* error bounded trajectory simplification) for large-scale trajectory data.

(1) We classify 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, which consist of optimal and sub-optimal algorithms, and the later is further classified into batch, online and one-pass algorithms. Note that online and one-pass algorithms are typically designed for a specific distance metric of PED, SED and DAD.

(2) Using four real-life trajectory datasets, we systematically evaluate and analyze the performance (compression ratio, average error and running time) of error bounded LS algorithms in terms of trajectory sizes and error bounds. (i) For a fair running time analysis, all algorithms are (re)-implemented in Java, unlike [41] that reports running time of algorithms with different programming languages. (ii) compression ratio analyses are not considered in [41]. (iii) Variations of distance metrics are not studied in [41]. (iv) Optimal algorithms using PED and SED and one-pass algorithms SIPED and CISED are not investigated in [41]. Indeed, algorithm SIPED is *completely overlooked* by existing trajectory compression studies as it is originally developed in fields of computational geometry and pattern recognition [10,34,39,42], and we find that it can be easily adopted for trajectory compression with good performance.

Essentially, this study is an important complimentary to [41] by providing a thorough and systematic evaluation and analyses of error bounded trajectory simplification algorithms. Further, though [41] tests the average errors of algorithms from the aspect of applications, *i.e.,* spatio-temporal queries, different applications have different requirements, and it is hardly to enumerate all of them. Hence, we focus on revealing the intrinsic characteristics of algorithms, and providing guidelines and suggestions on the choices of methods and distance metrics for different scenarios.

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 reports and analyzes the experimental results, followed by conclusions in Section 6.

2. PRELIMINARY

In this section, we introduce some basic concepts for trajectory simplification.

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 $0 \leq i < j \leq n$), where a data *point* is defined as a triple $P(x, y, t)$, which represents that a moving object is located at *longitude* x and *latitude* y at *time* t. Note that data points can be viewed as points in a three-dimension Euclidean space.

Piece-wise line representation. A *piece-wise line representation* $\overline{\mathcal{T}}[\mathcal{L}_0, \ldots, \mathcal{L}_m]$ $(0 < m \leq n)$ of a trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ is a sequence of continuous *directed line segments* (or line segment for simplicity) $\mathcal{L}_i = \overrightarrow{P_{s_i} P_{e_i}}$ $(i \in [0, m])$ 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 each directed line segment in $\overline{\mathcal{T}}$ essentially represents a continuous sequence of data points in $\ddot{\mathcal{T}}$.

For trajectory simplification, three distance metrics are commonly used, namely, the *perpendicular Euclidean distance* (PED), the *synchronous Euclidean distance* [22] (SED) and the *direction-aware distance* [19,41] (DAD). Consider a data point P and a directed line segment $\mathcal{L} = \overrightarrow{P_s P_e}$.

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

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

Direction-aware distance 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.,* $\Delta(\mathcal{L}_1.\theta, \mathcal{L}_2.\theta) = \min\{|\mathcal{L}_1.\theta - \mathcal{L}_2.\theta|, 2\pi - |\mathcal{L}_1.\theta - \mathcal{L}_2.\theta|\}$, where $\theta \in [0, 2\pi)$ is the angular of $\mathcal{L}$.

Min-# problem. Given a trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ and a pre-specified constant ϵ, the *min-#* problem of trajectory simplification is to approximate the trajectory $\ddot{\mathcal{T}}$ with $\overline{\mathcal{T}}[\mathcal{L}_0, \ldots, \mathcal{L}_m]$ $(0 < m \leq n)$, such that (1) on each of them the points $[P_{s_i}, \ldots, P_{e_i}]$ are approximated by a line segment $\mathcal{L}_i = \overrightarrow{P_{s_i} P_{e_i}}$ with the maximum PED or SED *error* of point P_j (or DAD *error* of line segment $|\overrightarrow{P_j P_{j+1}}|$) to line segment $\mathcal{L}_i$, $s_i \leq j < e_i$, less than ϵ, and (2) P_{s_i} and $P_{e_i} \in \ddot{\mathcal{T}}$.

3. OPTIMAL ALGORITHMS

Table 1: Error bounded trajectory simplification algorithms

Category		Algorithms	PED	SED	DAD	Time	Space	Rep	Key Ideas
optimal		Optimal [12]	✓	✓	✓	$O(n^3)$	$O(n^2)$	✓	Reachability Graph
		OptPED [6]	✓	×	×	$O(n^2)$	$O(n^2)$		Reachability Graph and Sector intersection
		SP [19]	×	×	✓	$O(n^2)$	$O(n^2)$		Reachability Graph and Range intersection
sub-optimal	batch	Ramer [29]	✓	✓	✓	$O(n^2)$	$O(n)$		Top-down
		DP [9, 22]	✓	✓	✓	$O(n^2)$	$O(n)$	✓	Top-down
		TP [27]	✓	✓	✓	$O(n^2/K)$	$O(n)$	✓	Bottom-up
	online	OPW [22]	✓	✓	✓	$O(n^2)$	$O(n)$	✓	Top-down and Opening window
		BQS [18]	✓	×	×	$O(n^2)$	$O(\|n\|)$	✓	Top-down, Opening window and Convex hull
		SWAB [14]	✓	✓	✓	$O(n * \|Q\|)$	$O(\|Q\|)$		Bottom-up and Sliding window
		SQUISH-E [25]	×	✓	×	$O(n \log \|Q\|)$	$O(\|Q\|)$	✓	Bottom-up and Priority queue
	one-pass	RW [30]	✓	×	×	$O(n)$	$O(1)$		Strip
		OPERB [17]	✓	×	×	$O(n)$	$O(1)$	✓	Fitting function
		SIPED [10, 42]	✓	×	×	$O(n)$	$O(1)$	✓	Sector intersection
		LDR [15, 37]	×	✓	×	$O(n)$	$O(1)$		Linear dead reckoning
		CISED [16]	×	✓	×	$O(n)$	$O(1)$	✓	Spatio-temporal Cone intersection
		Intersect [19]	×	×	✓	$O(n)$	$O(1)$	✓	Range intersection with $\frac{\epsilon}{2}$-range
		Interval [13]	×	×	✓	$O(n)$	$O(1)$	✓	Range intersection with ϵ-range

Here "Rep" means "Representative", K is the number of the final segments of a trajectory and $|Q|$ is the size of a buffer/window.

This section reviews the 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) [12] first formulates it as a graph reachability problem, and solves the problem in $O(n^3)$ time, where n is the number of the original points of a trajectory. It is initially designed to support PED, but can easily be modified to support SED and DAD. By using *convex hull* [36] and *sector intersection* [21], faster optimal algorithms are proposed with an improved time complexity to $O(n^2 \log n)$. Further, [6] presents an optimal algorithm using PED (OptPED) achieves $O(n^2)$ time by using the *sector intersection* mechanism, and algorithm SP [19] is essential an optimization of the original optimal algorithm using DAD that achieves $O(n^2)$ time. However, all the above optimization mechanisms do not support SED, and Optimal remains the best optimal solution for SED. *As all the optimized algorithms have the same effectiveness using the same distance metric, and essentially work for small size trajectories only, we choose algorithm* Optimal *as the representative of optimal* LS *algorithms.*

Given a trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ and an error bound ϵ, algorithm Optimal [12] 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 approximation 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 pair 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, *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.

4. SUB-OPTIMAL ALGORITHMS

This section elaborates the-state-of-the-art sub-optimal algorithms solving the *min-#* problem of trajectory simplification according to the taxonomy in 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 [22].

(1) Top-down algorithms recursively divide a trajectory into sub-trajectories until the stopping condition is met. Algorithms Ramer [29] and Douglas-Peucker (DP) [9] are similar, and support all the three distances PED, SED and DAD. An improved method of DP with a time complexity of $O(n \log n)$, based on *convex hulls*, is proposed in [11], which is the best DP based algorithm in terms of time complexities, and is designed for PED only, not for SED and DAD.

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

Note that, compared with top-down algorithms, bottom-up algorithms fit better for trajectories with lower sampling rates, as they typically need more rounds to merge smaller line segments into larger line segments. *Batch algorithms basically work for small and medium size trajectories, and*

we choose DP *and* TP *that all support* PED*,* SED *and* DAD *as the representatives of batch* LS *algorithms.*

Algorithm Douglas-Peucker (DP) [9]. 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.

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.

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

Given a trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ 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 $ped(P_{s+i}, \overrightarrow{P_s P_{s+k}})$ of each point P_{s+i} $(0 < i < k)$ to the line segment $\overrightarrow{P_s P_{s+k}}$, 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.

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 [22] is a combination of top-down algorithm DP and opening windows while SWAB [14] 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 [18]. To design more efficient online algorithms, techniques typically need to be designed closely coupled with distance metrics. Indeed, BQS [18] and SQUISH-E [25] propose to utilize convex hulls and priority queues, respectively, and they speed up trajectory simplification using PED and SED, respectively. To our knowledge, no specific techniques have been developed for DAD. Hence, *we choose algorithms* BQS*,* SQUISH-E *and* OPW *as the representatives of online algorithms using* PED*,* SED *and* DAD*, respectively.*

Algorithm BQS Using PED [18]. It is essentially an efficiency optimized OPW algorithm [22], and reduces the running time by introducing convex hulls to pick out a certain number of points, which makes it specific for PED.

For a buffer W with sub-trajectory $[P_s, \ldots, P_k]$, it splits the space into four quadrants. A buffer here is similar to a window in OPW [22]. 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 $\overrightarrow{P_s P_h}$ and $\overrightarrow{P_s P_l}$ have the largest and smallest angles with the x-axis, respectively. Here $P_h, P_l \in \{P_s, \ldots, P_k\}$. 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 $\overrightarrow{P_s P_k}$, among which the largest distance d_u and the smallest distance d_l are an upper bound and a lower bound of the distances of all points in $[P_s, \ldots, P_k]$ to line $\overrightarrow{P_s P_k}$. (1) If $d_l \geq \epsilon$, it produces a new line segment $\mathcal{L}(P_s, P_{k-1})$, and produces a new window $[P_{k-1}, \ldots, P_k]$ to replace W. (2) If $d_u < \epsilon$, it 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.

Algorithm SQUISH-E Using SED [25]. It is a bottom-up algorithm with a buffer, and has two forms: SQUISH-E (λ) ensuring the compression ratio λ, and SQUISH-E (ϵ) ensuring the SED error bound ϵ. In this study, we use 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 $P(pre, suc, prio, mnprio)$, where P is a trajectory data point, *pre* and *suc* are the predecessive and successive points of P, respectively, *prio* is the priority of P defined as an upper bound of the SED error that the removal of P introduces, and *mnprio* is the max priority of its predecessive and successive points removed from the list. Initially, trajectory points are loaded to Q one by one. At the same time, *mnprio* of each point is set to *zero* as no node has been removed from the list. Moreover, the priorities of points P_0 and $P_{|Q|-1}$ are set to ∞, and the priority of point P_i $(0 < i < |Q| - 1)$ is set to $sed(P_i, \overrightarrow{pre(P_i)suc(P_i)})$. 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.

4.3 One-pass Algorithms

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

Reumann-Witkam (RW) [30] is a straightforward one-pass

algorithm that builds a strip paralleling to the line connecting the first two points, then the points within this strip compose a section of the line. RW is fast, but has a poor compression ratio. Algorithm OPERB [17] recently improves RW by allowing dynamically adjustable strips, together with several detailed optimization techniques. There is also algorithm SIPED (*sector intersection*) that converts 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 [10,34,39,42]. These algorithms are 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 [15, 37] follows the similar routine as RW except that it uses SED and assumes a velocity $\vec{v}$ for each section. It has poor compression ratios because both the value and the direction of velocity $\vec{v}$ are pre-defined and fixed between two updates. Recently algorithm CISED [16] extends the *sector intersection* method SIPED from a 2D space to a Spatio-Temporal 3D space. These one-pass algorithms are designed for SED. *As algorithm* CISED *has a compression ratio close to algorithm* DP *using* SED*, we choose it as the representative of one-pass algorithms using* SED.

Direction range intersection approaches are similar to *sector intersection* methods except that they are designed for DAD, we choose Intersect [19] and Interval [13] as the representatives of one-pass algorithms using DAD.

Algorithm OPERB **Using** PED **[17]**. Consider an error bound ϵ and a sub-trajectory $\dddot{\mathcal{T}}_s[P_s, \ldots, P_{s+k}]$. 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 $\mathcal{R}_{i+1} = \overrightarrow{P_s P_{s+i+1}}$ as the global distance checking does. During processing, if the distance of point P_{s+i} to the directed line segment $\mathcal{L}_{i-1}$ is larger than the threshold, then a directed line segment, start from P_s, is generated and output; otherwise, the directed line segment $\mathcal{L}_i$ is updated by the fitting function $\mathbb{F}$, as follows.

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

where (a) $1 \le i \le k{+}1$; (b) $\mathcal{R}_{i-1} = \overrightarrow{P_s P_{s+i-1}}$, is the directed line segment whose end point P_{s+i-1} is in $\dddot{\mathcal{T}}_s[P_s, \ldots, P_{s+k}]$; (c) $\mathcal{L}_i$ is the directed line segment built by fitting function $\mathbb{F}$ to fit sub-trajectory $\dddot{\mathcal{T}}_s[P_s, \ldots, P_{s+i}]$ 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.

Algorithm SIPED **Using** PED **[10, 34, 39, 42]**. 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 $|\overrightarrow{P_s P_{s+i}}| > \epsilon$ $(i \in [1, k])$, there are two directed lines $\overrightarrow{P_s P^u_{s+i}}$ and $\overrightarrow{P_s P^l_{s+i}}$ such that $ped(P_{s+i}, \overrightarrow{P_s P^u_{s+i}}) = ped(P_{s+i}, \overrightarrow{P_s P^l_{s+i}}) = \epsilon$ and either $(\overrightarrow{P_s P^l_{s+i}}.\theta < \overrightarrow{P_s P^u_{s+i}}.\theta\ and\ \overrightarrow{P_s P^u_{s+i}}.\theta - \overrightarrow{P_s P^l_{s+i}}.\theta < \pi)$ or $(\overrightarrow{P_s P^l_{s+i}}.\theta > \overrightarrow{P_s P^u_{s+i}}.\theta\ and\ \overrightarrow{P_s P^u_{s+i}}.\theta - \overrightarrow{P_s P^l_{s+i}}.\theta < -\pi)$. Indeed, they form a *sector* $\mathcal{S}(P_s, P_{s+i}, \epsilon)$ that takes P_s as the center point and $\overrightarrow{P_s P^u_{s+i}}$ and $\overrightarrow{P_s P^l_{s+i}}$ as the border lines. There exists a data point Q such that for any data point P_{s+i} $(i \in [1, ...k])$, its perpendicular Euclidean distance to directed line $\overrightarrow{P_s Q}$ 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.,* $\bigcap_{i=1}^{k} \mathcal{S}(P_s, P_{s+i}, \epsilon) \neq \{P_s\}$ [34, 39, 42]. 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} $(1 \le i \le k)$ is chosen as Q, then for any point P_{s+j} $(j \in [1, ...i])$, its PED to line segment $\overline{P_s P_{s+i}}$ 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 [42]. That is, *these sector intersection based algorithms can be easily adopted for trajectory compression.*

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} $(j \in [1, ...i])$, its PED to line segment $\overline{P_s P_{s+i}}$ 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 preview full *sectors.*

Algorithm CISED **Using** SED **[16]**. Given a sub-trajectory $[P_s, ..., P_{s+k}]$ and an error bound ϵ, any point P'_{s+i} $(0 < i \le k)$ 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 $|P_{s+i}P'_{s+i}| \le \epsilon$, they form a *synchronous circle* $\mathcal{O}(P_{s+i}, \epsilon)$ 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} $(1 \le i \le k)$ in $\dddot{\mathcal{T}}_s$ *w.r.t.* a point P_s and an error bound ϵ, denoted as $\mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon))$, or $\mathcal{C}_{s+i}$ in short, is an oblique circular cone such that point P_s is its apex and the synchronous circle $\mathcal{O}(P_{s+i}, \epsilon)$ is its base. Then, there exists a point Q such that $Q.t = P_{s+k}.t$ and $sed(P_{s+i}, \overrightarrow{P_s Q}) \le \epsilon$ for each $i \in [1, k]$ if and only if $\bigcap_{i=1}^{k} \mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon)) \neq \{P_s\}$. Like *sector intersection* methods, point Q may also not belong to $\{P_s, P_{s+1}, \ldots, P_{s+k}\}$. If P_{s+i} $(1 \le i \le k)$ is chosen as Q, then for any point P_{s+j} $(j \in [1, ...i])$, its SED to line segment $\overline{P_s P_{s+i}}$ is not greater than the error bound ϵ if $\bigcap_{j=1}^{i} \mathcal{C}(P_s, P_{s+j}, \epsilon/2) \neq \{P_s\}$ as pointed out in [16].

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 intersection of cone projection circles on a plane, and a circle is further approximated with its m-edge inscribed regular polygon, whose intersection can be computed more efficiently.

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

Algorithms Intersect **[19] and** Interval **[13] Using** DAD**.** Given a direction line segment $\mathcal{L}$ and an angle

Table 2: Real-life trajectory datasets

Data Sets	Number of Trajectories	Sampling Rates (s)	Points Per Trajectory	Total points
Taxi	500	60	$\sim$ 42.8K	21.4M
UCar	200	3-5	$\sim$ 114.0K	22.8M
Geolife	182	1-5	$\sim$ 131.4K	24.2M
Mopsi	51	2	$\sim$ 153.9K	7.9M

ϵ, the *direction range* denoted by $range(\mathcal{L}.\theta, \epsilon)$ is $[\mathcal{L}.\theta - \epsilon, \mathcal{L}.\theta + \epsilon]$, where $[\theta_1, \theta_2]$ 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 [19]. 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 [19] uses a half range, and shows that if the common intersection $\bigcap_{i=1}^{k} range(\overrightarrow{P_{s+i-1}P_{s+i}}.\theta, \epsilon/2) \neq \phi$, then the angular between $\overrightarrow{P_{s+i-1}P_{s+i}}$ and $\overrightarrow{P_sP_{s+k}}$ for all $i \in [1, k]$ is not larger than ϵ. Recently, algorithm Interval [13] extends Intersect from half to full ranges, by showing that if the common intersection $\bigcap_{i=1}^{k} range(\overrightarrow{P_{s+i-1}P_{s+i}}.\theta, \epsilon) \neq \phi$ and $\overrightarrow{P_sP_{s+k}}.\theta$ falls in the common intersection, then the angular between $\overrightarrow{P_{s+i-1}P_{s+i}}$ and $\overrightarrow{P_sP_{s+k}}$ for all $i \in [1, k]$ is not larger than ϵ.

5. EVALUATION

In this section, we present extensive and systematic experimental studies and analyses of eleven representative LS algorithms. Using four real-life datasets, we conduct three sets of tests to evaluate compression ratios, errors and efficiency of these representative algorithms using distance metrics PED, SED and DAD, and the impacts of error bounds ϵ and trajectory sizes.

5.1 Experimental Setting

Real-life Trajectory Datasets. We use four varied real-life datasets shown in Table 2, namely, taxi trajectory data (Taxi) collected by a Beijing taxi company, Service car trajectory data (UCar) collected by a Chinese car rental company, Geolife trajectory data (Geolife) collected in GeoLife project [1] and Mopsi trajectory data (Mopsi) collected in Mopsi project [2], to evaluate those LS algorithms. These data sets have varied sampling rates, ranging from one point per minute to one point per second. They also come from different sources, where Taxi and UCar are collected by cars in urban, and Geolife and Mopsi are a mixing of cars and individuals. The data source and sampling rate also affect the performance of LS algorithms using certain distance metrics.

Algorithms and implementation. We implement the representative algorithms of Table 1. They are optimal algorithm Optimal, batch algorithms DP and TP, online algorithms OPW, BQS and SQUISH-E, and one-pass algorithms OPERB, SIPED, CISED, Intersect and Interval. For one-pass algorithms SIPED and CISED, we implement two versions of them, *i.e.,* the half and full ranges for algorithm CISED, a fixed parameter $m = 16$ as evaluated in [16], *i.e.,* 16-edges inscribe regular polygon. All algorithms were implemented with Java. All tests were 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 varied 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 chose 10 trajectories from each dataset, and varied the size $|\ddot{\mathcal{T}}|$ of a trajectory from $1,000$ points to $10,000$ points while fixed the error bound $\epsilon = 40$ metres or $\epsilon = 45$ degrees.

5.2 Evaluation Metrics

Compression ratios, errors and running time are the most popular metrics to evaluate LS algorithms.

Compression ratios. For trajectories $\{\ddot{\mathcal{T}}_1, \ldots, \ddot{\mathcal{T}}_M\}$ and their piecewise 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.

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

Running time. It is the efficicency of algorithms.

5.3 Experimental Results and Analysese

We next present our findings.

5.3.1 *Compression Ratio Evaluation and Analyses*

The compression ratios of these algorithms under varied error bounds ϵ and trajectory sizes are reported in Figures 1, 2, 3, 4, 5 and 6. Note that the optimal algorithm using SED and DAD is not reported in Figures 1, 2 and 3 as it runs out of memory when compressing the full dataset. We first report our findings.

(1) The compression ratios of algorithms using PED from the best to the worst are the Optimal algorithm, online algorithm BQS, one-pass algorithm SIPED using a full ϵ sector (SIPED (ϵ) in short), batch algorithms TP and DP, and one-pass algorithms SIPED ($\frac{\epsilon}{2}$) and OPERB. The output sizes of algorithms BQS and SIPED (ϵ) are on average (103.58%, 113.32%, 120.22%, 120.83%) and (103.98%, 116.04%, 124.46%, 124.24%) of the optimal algorithm Optimal on datasets (Taxi, UCar, Geolife, Mopsi), respectively. Algorithms TP and DP are comparable, and their output sizes are on average (103.17%, 125.05%, 131.01%, 138.01%) and (106.98%, 130.03%, 140.56%, 139.00%) of Optimal on (Taxi, UCar, Geolife, Mopsi), respectively. Algorithms SIPED ($\frac{\epsilon}{2}$) and OPERB are comparable, and they are on average (113.09%, 136.73%, 150.23%, 152.29%) and (119.89%, 143.14%, 147.80%, 152.37%) of Optimal on (Taxi, UCar, Geolife, Mopsi), respectively. For example, in Mopsi, the compression ratios of algorithms (Optimal, TP, DP, BQS, SIPED (ϵ), SIPED ($\frac{\epsilon}{2}$), OPERB) are (1.6%, 2.2%, 2.2%, 1.9%, 2.0%, 2.4%, 2.4%) when $\epsilon = 40m$.

(2) The compression ratios of algorithms using SED from the best to the worst are the Optimal algorithm, one-pass algorithm CISED (ϵ), batch algorithms TP and DP, one-pass algorithm CISED ($\frac{\epsilon}{2}$), and online algorithm SQUISH-E. Algorithms TP and DP are comparable, and they are on average (102.72%, 125.23%, 143.92%, 128.63%) and (103.18%, 123.93%, 141.46%, 121.14%) of algorithm Optimal on datasets (Taxi, UCar, Geolife, Mopsi), respectively. Algorithms CISED (ϵ), CISED ($\frac{\epsilon}{2}$) and SQUISH-E

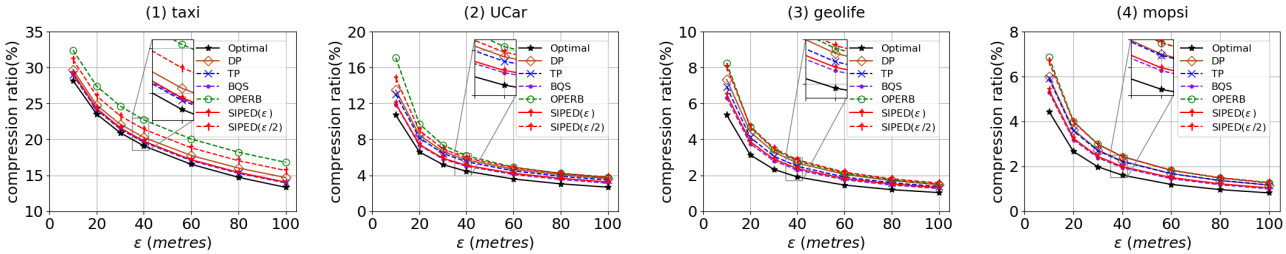

Figure 1: Evaluation of compression ratios (PED) on full datasets: varying the error bound ϵ.

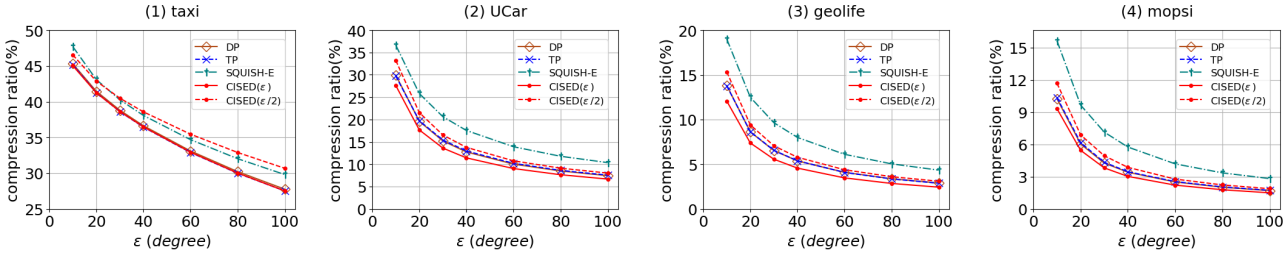

Figure 2: Evaluation of compression ratios (SED) on full datasets: varying the error bound ϵ.

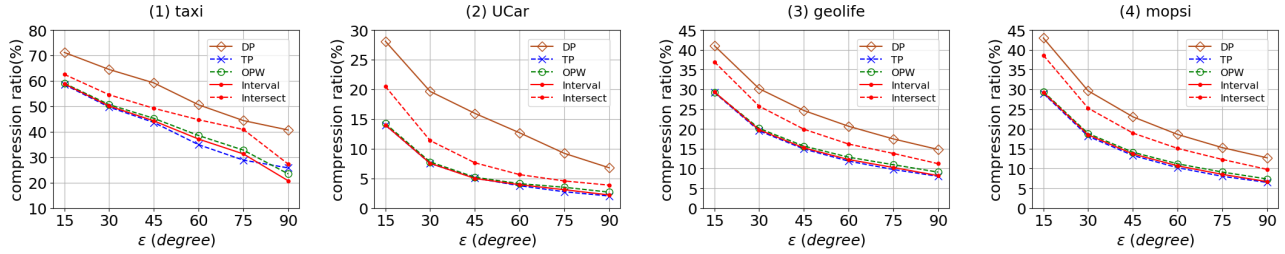

Figure 3: Evaluation of compression ratios (DAD) on full datasets: varying the error bound ϵ.

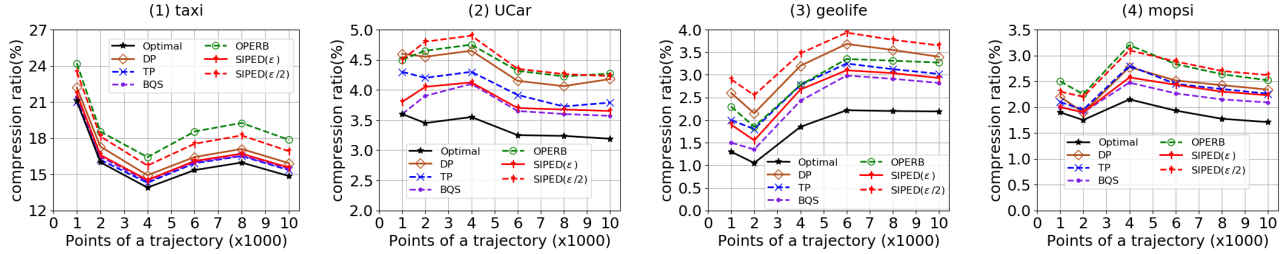

Figure 4: Evaluation of compression ratios (PED) on small datasets: varying the size of trajectories.

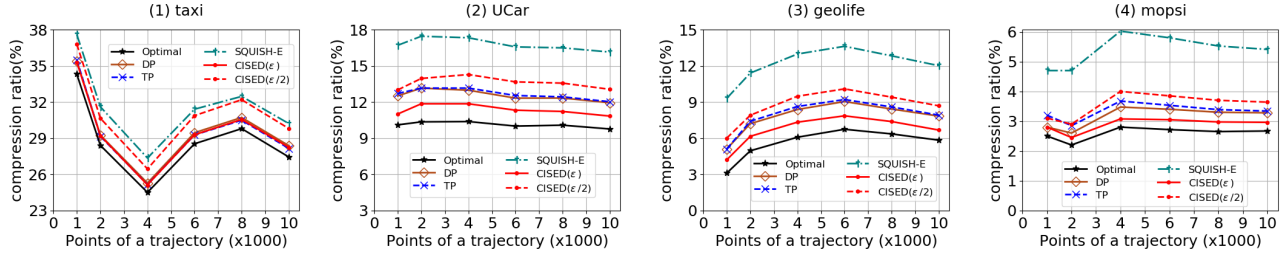

Figure 5: Evaluation of compression ratios (SED) on small datasets: varying the size of trajectories.

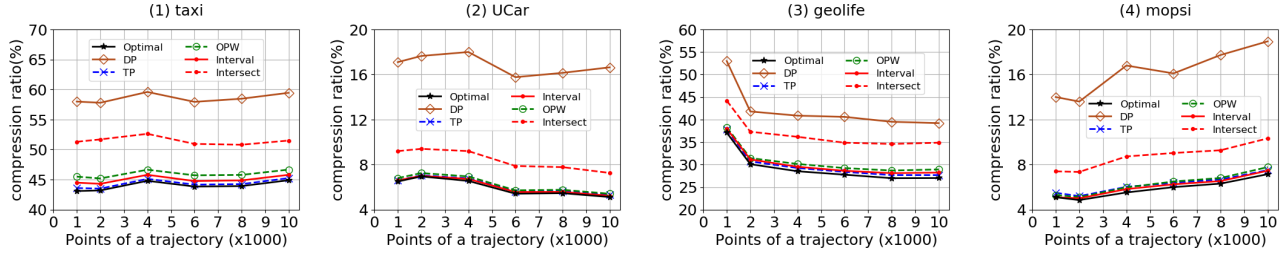

Figure 6: Evaluation of compression ratios (DAD) on small datasets: varying the size of trajectories.

are on average (102.04%, 109.27%, 110.13%, 115.90%,), (108.00%, 134.35%, 159.30%, 136.06%) and (110.27%, 165.94%, 225.68%, 206.90%) of Optimal on (Taxi, UCar, Geolife, Mopsi), respectively. For example, in Mopsi, the compression ratios of algorithms (TP, DP, SQUISH-E, CISED (ϵ), CISED ($\frac{\epsilon}{2}$)) are (3.45%, 3.41%, 5.75%, 3.02%, 3.86%), respectively, when $\epsilon = 40m$.

(3) 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 (100.81%, 102.91%, 102.27%, 106.88%), (104.39%, 116.09%, 107.11%, 115.42%) and (102.38%, 101.98%, 103.52%, 103.43%) of algorithm Optimal on datasets (Taxi, UCar, Geolife, Mopsi), respectively. Algorithms Intersect and DP are on average (111.72%, 156.00%, 121.20%, 230.52%) and (133.19%, 283.93%, 143.79%, 278.89%) of algorithm Optimal on datasets (Taxi, UCar, Geolife, Mopsi), respectively. For example, in Mopsi, the compression ratios of algorithms (TP, DP, OPW, Interval, Intersect) are (13.3%, 23.1%, 14.12%, 13.7%, 18.96%), respectively, when $\epsilon = 45$ degrees.

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

Analyses of LS algorithms. The Optimal algorithm is the best in term 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" [24]. Its "prediction" method is not accurate enough, thus, in order to ensure the error bound, it may ignore too many potential points that could be represented by a line segment.

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

Analyses of distance metrics. Though PED, SED and DAD are different distances, the comparison of their compression ratios is helpful to choose an effective distance metric. First, given the same error bound ϵ, the compression ratios of algorithms using PED are obviously better than using SED. More specifically, *the output sizes of using* SED *are approximately twice of* PED. As shown in Figures 1 and 2, the output sizes of algorithms TP and DP using PED are on average (55.08%, 43.55%, 47.49%, 63.15%) and (56.75%, 45.79%, 50.88%, 64.50%) of algorithms TP and DP using SED on datasets (Taxi, UCar, Geolife, Mopsi), respectively. This result shows SED saves temporal information at a price with twice more points.

Secondly, in practice (*e.g.,* $\epsilon < 100$ meters and $\epsilon < 60$ degrees), SED have obviously better compression ratios than DAD in datasets Geolife and Mopsi, a bit better than DAD in Taxi 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 Taxi and UCar, which capture more direction changes, *i.e.,* direction changes in a small time interval.

5.3.2 Average Error Evaluation and Analyses

The average errors of these algorithms under varied error bounds ϵ and trajectory sizes are reported in Figures 7, 8, 9, 10, 11 and 12. We first report our findings.

(1) When using PED, the average errors from the smallest to the largest are batch algorithms TP and DP, one-pass algorithms SIPED ($\frac{\epsilon}{2}$) and OPERB, the Optimal algorithm and one-pass algorithm SIPED (ϵ), and online algorithm BQS. For full datasets, algorithms TP and DP are comparable, and they are on average (77.43%, 58.69%, 61.34%, 57.57%) and (88.12%, 57.61%, 62.66%, 60.23%) of Optimal on datasets (Taxi, UCar, Geolife, Mopsi), respectively. Algorithms SIPED ($\frac{\epsilon}{2}$) and OPERB are comparable, and they are on average (73.08%, 80.96%, 79.12%, 79.33%), (73.03%, 70.60%, 76.64%, 78.71%) of Optimal on datasets (Taxi, UCar, Geolife, Mopsi), respectively. Algorithms SIPED (ϵ) and BQS are on average (87.63%, 100.05%, 101.01%, 102.69%) and (97.69%, 104.67%, 108.91%, 106.92%) of Optimal on datasets (Taxi, UCar, Geolife, Mopsi), respectively. For example, the average errors of algorithms (Optimal, TP, DP, BQS, SIPED (ϵ), SIPED ($\frac{\epsilon}{2}$), OPERB) in the full Mopsi are (16.08, 9.19, 9.68, 17.4, 12.96, 16.83, 12.77) metres when $\epsilon = 40m$.

(2) When using SED, the average errors from the smallest to the largest are online algorithm SQUISH-E, batch algorithms TP and DP, one-pass algorithm CISED ($\frac{\epsilon}{2}$), and one-pass algorithms CISED (ϵ) and the Optimal algorithm. Algorithms TP and DP are comparable, and they are on average (87.22%, 60.36%, 66.11%, 62.43%), (81.04%, 62.54%,

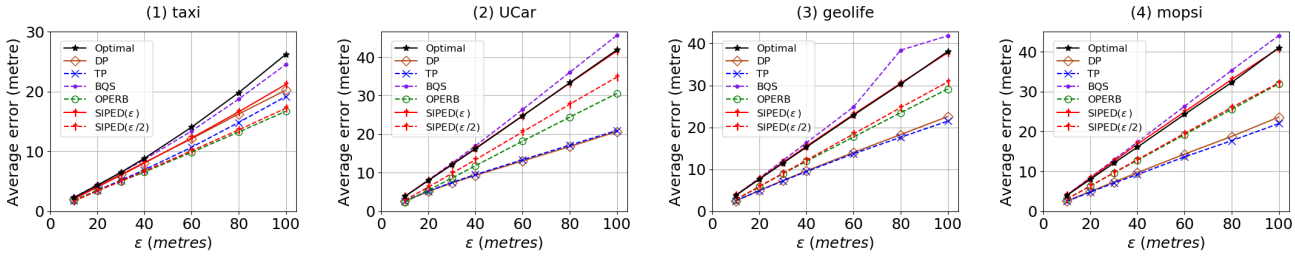

Figure 7: Evaluation of average errors (PED) on full datasets: varying the error bound ϵ.

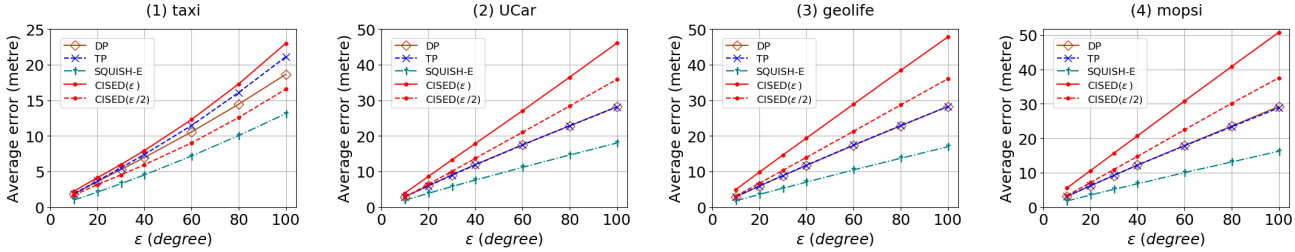

Figure 8: Evaluation of average errors (SED) on full datasets: varying the error bound ϵ.

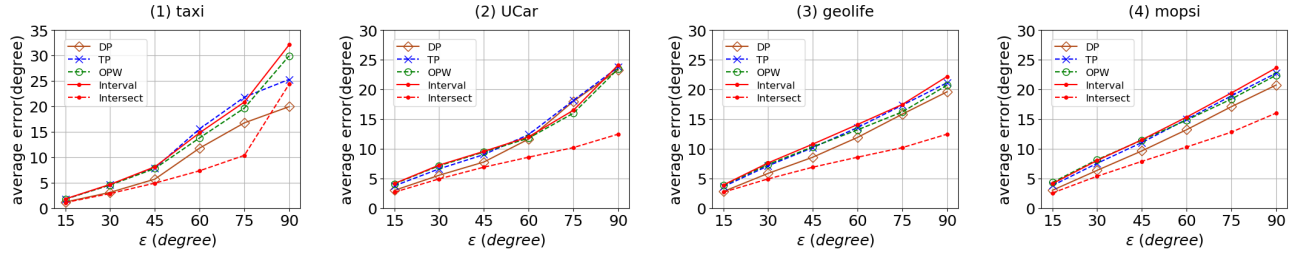

Figure 9: Evaluation of average errors (DAD) on full datasets: varying the error bound ϵ.

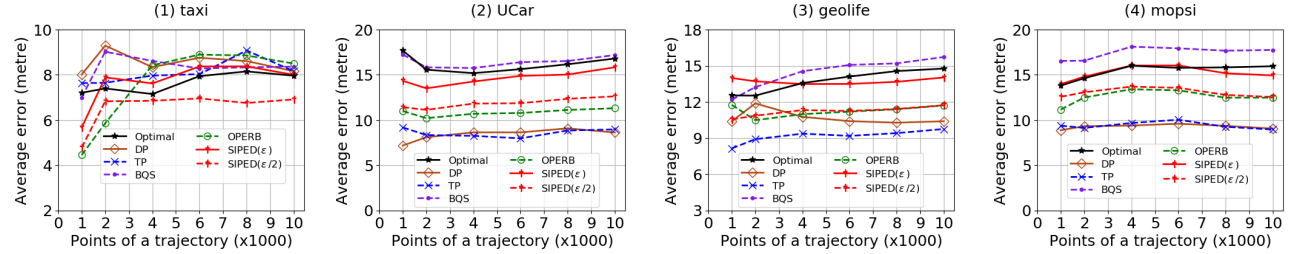

Figure 10: Evaluation of average errors (PED) on small datasets: varying the size of trajectories.

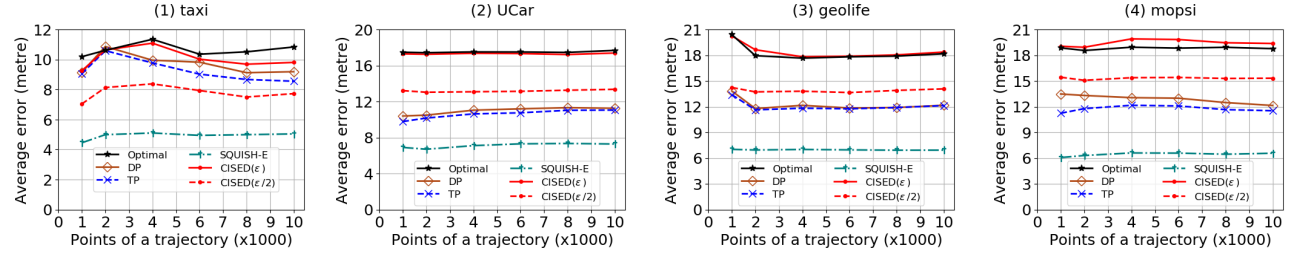

Figure 11: Evaluation of average errors (SED) on small datasets: varying the size of trajectories.

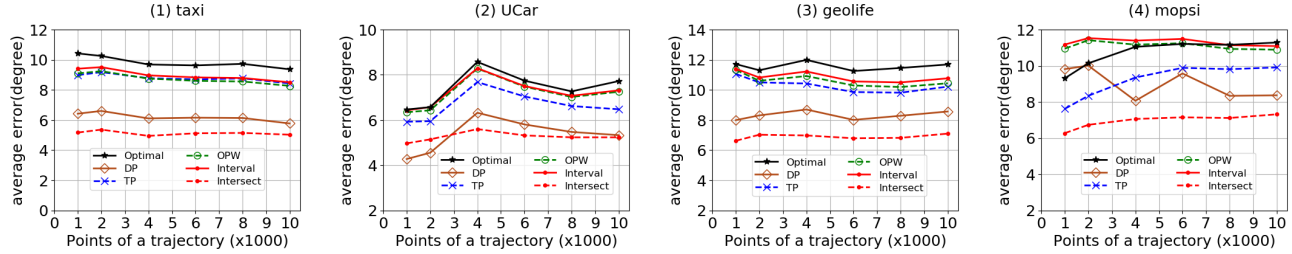

Figure 12: Evaluation of average errors (DAD) on small datasets: varying the size of trajectories.

67.04%, 68.64%) of Optimal on datasets (Taxi, UCar, Geolife, Mopsi), respectively. Algorithms CISED (ϵ), CISED ($\frac{\epsilon}{2}$) and SQUISH-E are on average (95.07%, 97.32%, 106.74%, 108.16%), (73.15%, 75.29%, 76.03%, 81.44%) and (46.27%, 40.61%, 38.15%, 34.22%) of Optimal on datasets (Taxi, UCar, Geolife, Mopsi), respectively. For example, the average errors of algorithms (Optimal, TP, DP, SQUISH-E, CISED (ϵ), CISED ($\frac{\epsilon}{2}$)) in full Mopsi are (19.39, 12.17, 12.20, 6.76, 20.68, 14.71) metres, respectively, when $\epsilon = 40m$.

(3) 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. Algorithms TP, OPW and Interval are comparable, and they are on average (88.94%, 91.35%, 61.45%, 73.71%), (88.23%, 91.95%, 61.37%, 76.17%) and (93.97%, 90.36%, 68.23%, 163.47%) of Optimal on datasets (Taxi, UCar, Geolife, Mopsi), respectively. Algorithms Intersect and DP are on average (66.17%, 62.03%, 76.54%, 110.69%) and (76.86%, 82.45%, 96.52%, 137.95%) of Optimal on datasets (Taxi, 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 term of average errors, followed by one-pass algorithms and then batch algorithms. Online algorithms have varied average errors, ranging from the best to the worst. (1) For batch algorithms, both bottom-up algorithm (TP) and top-down algorithm (DP) have similar average errors, and they are pretty good compared with other algorithms. (2) Online algorithms BQS and OPW often have the largest average errors in all sub-optimal algorithms, while SQUISH-E has the smallest. This is also on the contrary of their compression ratios. (3) For one-pass algorithms, the full ϵ sector/cone/range combining with a position/direction constraint always have larger average errors than the half ϵ sector/cone/range. Local distance checking approaches try to include more points into a line segment, this greedy strategy is likely leading to larger average errors, considerable larger than batch algorithms that have the similar compression ratios as one-pass and online algorithms.

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

5.3.3 Efficiency Evaluation and Analyses

We finally compare the efficiency of these algorithms. The results are reported in Figures 13, 14, 15, 16, 17 and 18. 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) When using PED, in most cases, the running time from the smallest to the largest is one-pass algorithms SIPED and OPERB, batch algorithms TP and DP, and online algorithm BQS. Algorithms SIPED ($\frac{\epsilon}{2}$) and OPERB are comparable, algorithm SIPED (ϵ) is $(0.99, 0.92, 0.92, 0.91)$ times of SIPED ($\frac{\epsilon}{2}$), and algorithms TP, DP and BQS are on average (19.19, 26.79, 28.25, 29.87), (17.90, 16.32, 15.40, 11.02) and (15.07, 37.73, 62.23, 61.29) times slower than one-pass algorithm SIPED ($\frac{\epsilon}{2}$) on datasets (Taxi, UCar, Geolife, Mopsi), respectively. For example, in Mopsi, the running time of algorithms (TP, DP, BQS, SIPED (ϵ), SIPED ($\frac{\epsilon}{2}$), OPERB) is (232.9, 124.2, 469.4, 6.89, 7.6, 8.6) seconds when $\epsilon = 40m$.

(2) When using SED, the running time from the smallest to the largest is one-pass algorithm CISED, online algorithm SQUISH-E, and batch algorithms TP and DP. Algorithm CISED (ϵ) is $(1.17, 1.17, 1.17, 0.91)$ times of CISED ($\frac{\epsilon}{2}$), and algorithms TP, DP and SQUISH-E are on average (8.58, 13.33, 15.81, 13.09), (12.81, 12.93, 10.64, 8.79) and (2.63, 2.75, 2.78, 2.57) times slower than CISED ($\frac{\epsilon}{2}$) on datasets (Taxi, UCar, Geolife, Mopsi), respectively. For example, in Mopsi, the running time of algorithms (TP, DP, SQUISH-E, CISED (ϵ), CISED ($\frac{\epsilon}{2}$)) is (156.6, 104.8, 27.2, 11.6, 9.7) seconds when $\epsilon = 40m$.

(3) When using DAD, the running time from the smallest to the largest is one-pass algorithms Intersect and Interval, batch algorithms TP and DP, and online algorithm OPW. Algorithm Interval is $(1.86, 1.80, 1.84, 1.81)$ times slower than Intersect, and algorithms TP, DP and OPW are on average (12.53, 24.63, 23.53, 23.23), (21.19, 25.49, 30.11, 31.72) and (6.84, 39.29, 147.85, 80.09) times slower than Intersect on datasets (Taxi, 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 algorithms DP and TP decreases or increases with the increase of error bound ϵ, respectively, due to the top-down and bottom-up approaches that they apply. When using PED or SED, top-down algorithm usually runs faster than bottom-up algorithm when the error bound ϵ is large (*e.g.,* in Geolife, $\epsilon > 10$ metres when using PED and $\epsilon > 30$ metres when using SED), which means that top-down (bottom-up) algorithm needs to split (merge) the original trajectory fewer (more) times in these cases, vice versa. When using DAD, top-down algorithms are normally a bit slower than bottom-up algorithms (recall that top-down algorithms have poorer compression ratios compared with bottom-up algorithms, which means that it needs more time to split the raw trajectory into more sub trajectories). In addition to error bounds, sampling rates also have impacts on the efficiency of batch algorithms. A dataset with high sampling rate likely needs more merging processes than splitting processes, thus, top-down algorithms run faster than bottom-up algorithms in

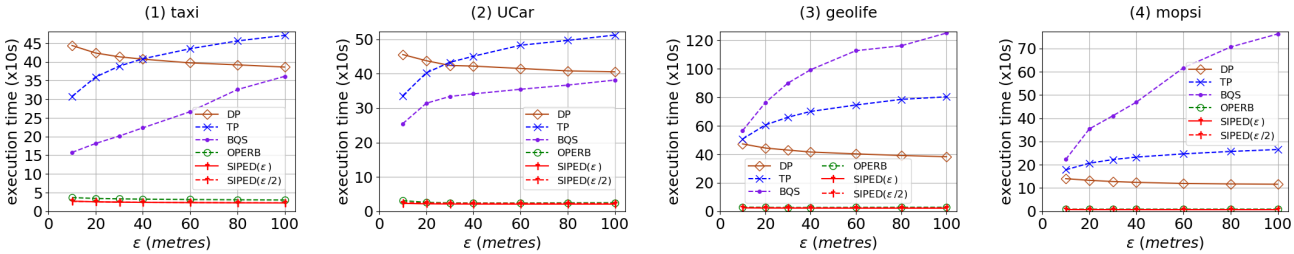

Figure 13: Evaluation of running time (PED) on full datasets: varying the error bound ϵ.

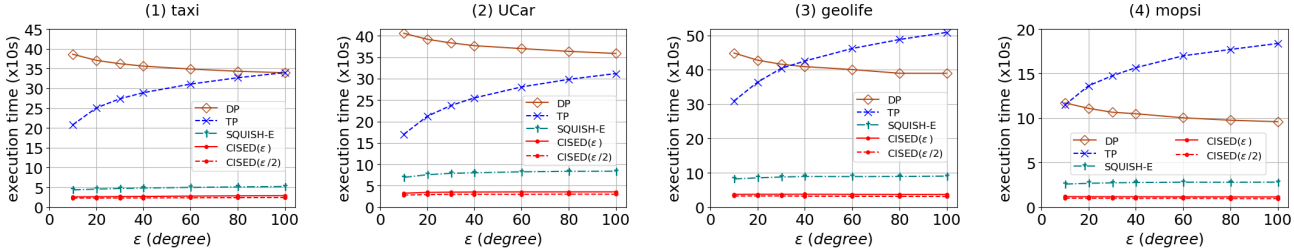

Figure 14: Evaluation of running time (SED) on full datasets: varying the error bound ϵ.

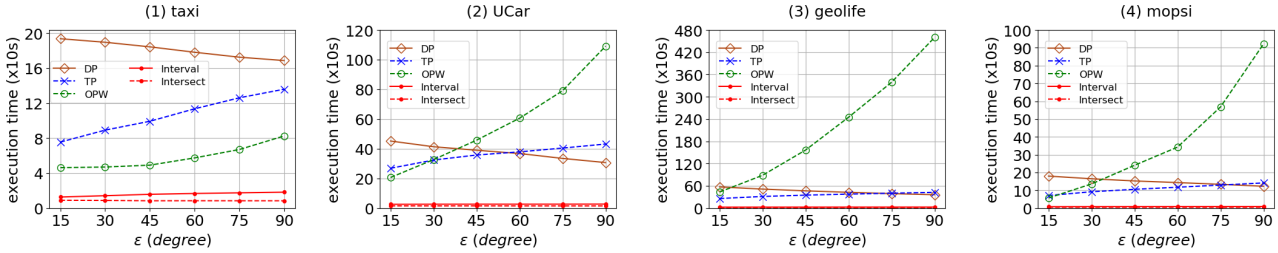

Figure 15: Evaluation of running time (DAD) on full datasets: varying the error bound ϵ.

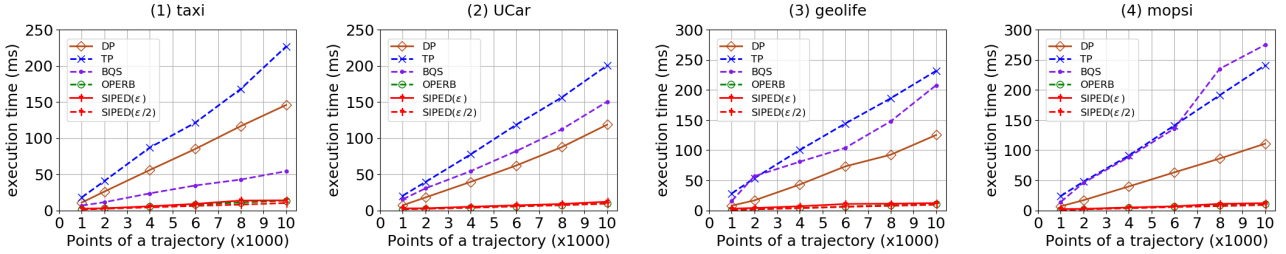

Figure 16: Evaluation of running time (PED) on small datasets: varying the size of trajectories.

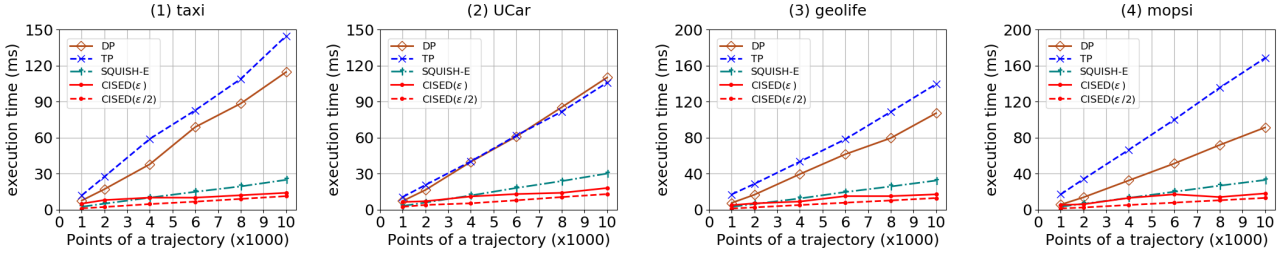

Figure 17: Evaluation of running time (SED) on small datasets: varying the size of trajectories.

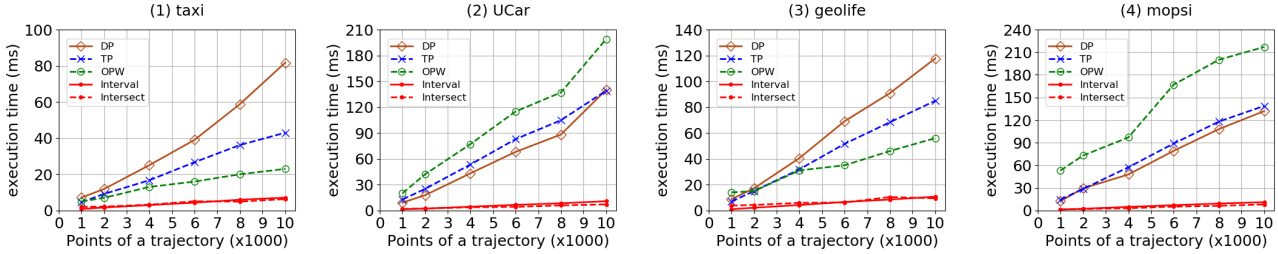

Figure 18: Evaluation of running time (DAD) on small datasets: varying the size of trajectories.

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, and batch approaches running in a buffer are still time consuming.

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

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

5.3.4 Summary

From these tests we find the followings.

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

Distance Metrics. 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. The average errors of algorithms using SED are a bit larger than using PED. Using DAD is in general faster than using PED and SED, and, indeed, the computation time of PED and SED is 2.3 and 1.7 times of DAD, respectively.

6. CONCLUSIONS

Using four real-life trajectory datasets, we have systematically evaluated and analyzed error bounded LS algorithms for trajectory compression, including *both the optimal and the sub-optimal methods that use* PED*,* SED *and/or* DAD, in terms of compression ratios, average errors and efficiency. Our experimental study and analyses show that:

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

For sub-optimal algorithms, the output sizes of algorithms BQS and SIPED (ϵ) using PED, CISED (ϵ) using SED, and TP and Interval using DAD are approximately 103%–124%, 102%–115% and 102%–107% of the optimal algorithms, respectively. Batch algorithms using PED and SED also have good compression ratios. Thus, they are the alternates of the optimal algorithms. More specifically, in case compression ratios are the first consideration, then algorithms BQS and SIPED (ϵ) using PED, CISED (ϵ) using SED, and TP and Interval using DAD are good candidates. In case average errors are concerned, then batch algorithms are good candidates as one-pass algorithms and online algorithms OPW and BQS all have large average errors. In case running time is the important factor or computing resources are limited, then one-pass algorithms are the best candidates. Indeed, one-pass algorithms run fast and require few resources, and have comparable compression ratios compared with batch and online algorithms. Hence, they are prominent trajectory compression algorithms when average errors are not the main concern.

(2) Choice of distance metrics. Users essentially choose a distance metric of PED, SED and DAD from the needs of applications. Further, the choice of a distance metric has impacts on the performance. For compression ratios, the using of synchronized distance SED saves temporal information of trajectories with a cost of approximately double-sized outputs compared with using PED in all datasets; PED has obvious better compression ratios than DAD in all datasets and SED is also better than DAD in high sampling datasets. For efficiency, the computation time of PED and SED are 2.3 and 1.7 times of DAD, respectively.

7. REFERENCES

[1] Geolife gps trajectory, https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset/.

[2] Mopsi routes, http://cs.uef.fi/mopsi/routes/dataset/.

[3] P. K. Agarwal and K. R. Varadarajan. Efficient algorithms for approximating polygonal chains. *Discrete and Computational Geometry*, 23:273–291, 2000.
[4] H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *VLDBJ*, 15(3):211–228, 2006.
[5] W. Cao and Y. Li. Dots: An online and near-optimal trajectory simplification algorithm. *Journal of Systems and Software*, 126(Supplement C):34–44, 2017.
[6] W. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimal error. *IJCGA*, 6(1):378–387, 1996.
[7] M. Chen, M. Xu, and P. Fränti. A fast multiresolution polygonal approximation algorithm for GPS trajectory simplification. *TIP*, 21(5):2770–2785, 2012.
[8] Y. Chen, K. Jiang, Y. Zheng, C. Li, and N. Yu. Trajectory simplification method for location-based social networking services. In *LBSN*, 2009.
[9] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.
[10] J. G. Dunham. Piecewise linear approximation of planar curves. *PAMI*, 8, 1986.
[11] J. Hershberger and J. Snoeyink. Speeding up the douglas-peucker line-simplification algorithm. *Technical Report, University of British Columbia*, 1992.
[12] H. Imai and M. Iri. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics, and Image Processing*, 36:31–41, 1986.
[13] B. Ke, J. Shao, and D. Zhang. An efficient online approach for direction-preserving trajectory simplification with interval bounds. In *MDM*, 2017.
[14] E. J. Keogh, S. Chu, D. M. Hart, and M. J. Pazzani. An online algorithm for segmenting time series. In *ICDM*, 2001.
[15] R. Lange, F. Dürr, and K. Rothermel. Efficient real-time trajectory tracking. *VLDBJ*, 20(5):671–694, 2011.
[16] X. Lin, J. Jiang, S. Ma, Y. Zuo, and C. Hu. One-pass trajectory simplification using the synchronous euclidean distance. *arXiv*, 2017.
[17] X. Lin, S. Ma, H. Zhang, T. Wo, and J. Huai. One-pass error bounded trajectory simplification. *PVLDB*, 10(7):841–852, 2017.
[18] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, and R. Jurdak. Bounded quadrant system: Error-bounded trajectory compression on the go. In *ICDE*, 2015.
[19] C. Long, R. C.-W. Wong, and H. Jagadish. Direction-preserving trajectory simplification. *PVLDB*, 6(10):949–960, 2013.
[20] C. Long, R. C.-W. Wong, and H. Jagadish. Trajectory simplification: On minimizing the direction-based error. *PVLDB*, 8(1):49–60, 2014.
[21] A. Melkman and J. O'Rourke. On polygonal chain approximation. *Machine Intelligence and Pattern Recognition*, 6:87–95, 1988.
[22] N. Meratnia and R. A. de By. Spatiotemporal compression techniques for moving point objects. In *EDBT*, 2004.
[23] J. Muckell, J. Hwang, C. T. Lawson, and S. S. Ravi. Algorithms for compressing GPS trajectory data: an empirical evaluation. In *ACM-GIS*, 2010.
[24] J. Muckell, J. Hwang, V. Patil, C. T. Lawson, F. Ping, and S. S. Ravi. SQUISH: an online approach for GPS trajectory compression. In *COM.Geo*, 2011.
[25] J. Muckell, P. W. Olsen, J.-H. Hwang, C. T. Lawson, and S. S. Ravi. Compression of trajectory data: a comprehensive evaluation and new approach. *GeoInformatica*, 18(3):435–460, 2014.
[26] A. Nibali and Z. He. Trajic: An effective compression system for trajectory data. *TKDE*, 27(11):3138–3151, 2015.
[27] T. Pavlidis and S. L. Horowitz. Segmentation of plane curves. *TOC*, 23(8):860–870, 1974.
[28] I. S. Popa, K. Zeitouni, VincentOria, and A. Kharrat. Spatio-temporal compression of trajectories in road networks. *GeoInformatica*, 19(1):117–145, 2014.
[29] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1:244–256, 1972.
[30] K. Reumann and A. Witkam. Optimizing curve segmentation in computer graphics. In *International Computing Symposium*, 1974.
[31] K.-F. Richter, F. Schmid, and P. Laube. Semantic trajectory compression: Representing urban movement in a nutshell. *JOSIS*, 4(1):3–30, 2012.
[32] F. Schmid, K. Richter, and P. Laube. Semantic trajectory compression. In *SSTD*, 2009.
[33] W. Shi and C. Cheung. Performance evaluation of line simplification algorithms for vector generalization. *Cartographic Journal*, 43(1):27–44, 2006.
[34] J. Sklansky and V. Gonzalez. Fast polygonal approximation of digitized curves. *Pattern Recognition*, 12:327–331, 1980.
[35] R. Song, W. Sun, B. Zheng, and Y. Zheng. Press: A novel framework of trajectory compression in road networks. *PVLDB*, 7(9):661–672, 2014.
[36] G. T. Toussaint. On the complexity of approximating polygonal curves in the plane. In *International Symposium on Robotics and Automation*, 1985.
[37] G. Trajcevski, H. Cao, P. Scheuermanny, O. Wolfsonz, and D. Vaccaro. On-line data reduction and the quality of history in moving objects databases. In *MobiDE*, 2006.
[38] R. van Hunnik. Extensive comparison of trajectory simplification algorithms. master thesis, 2017.
[39] C. M. Williams. An efficient algorithm for the piecewise linear approximation of planar curves. *Computer Graphics and Image Processing*, 8:286–293, 1978.
[40] F. Wu, K. Fu, Y. Wang, and Z. Xiao. A graph-based minimal number and error-optimal trajectory simplification algorithm and its extension towards online services. *ISPRS International Journal of Geo-Information*, 6(19), 2017.
[41] D. Zhang, M. Ding, D. Yang, Y. Liu, J. Fan, and H. T. Shen. Trajectory simplification: An experimental study and quality analysis. *PVLDB*, 9(11):934–946, 2018.
[42] Z. Zhao and A. Saalfeld. Linear-time sleeve-fitting polyline simplification algorithms. In *AutoCarto*, 1997.

Appendix A: Additional Related Work

Apart from the techniques evaluated in this paper, other approaches that satisfy different requirements, have been proposed or could be applied for trajectory compression. We next summarized other representative LS algorithms. Interested readers refer to [16, 25, 33, 41] for more information.

Weak simplification algorithms. The weak simplification allows that the output data points may not belong to the original data sets [37] (otherwise, the strong simplification). That is, weak simplification allows data interpolation. Algorithms Sleeve [42], OPERB [17] and CISED [16] are both strong and weak LS algorithms whose weak simplification versions have better compression ratios than their strong simplification versions, and even comparable with the optimal algorithms [16].

Algorithms using other error metrics. A number of algorithms [3,5,7,40] have been developed to solve the "min-#" problem under alternative error metrics. [3] studied the problem under the L_1 and uniform (also known as Chebyshev) metric. [7] defined the integral square synchronous Euclidean distance (ISSED), and used it to speed up the construction of reachability graphs, [40] followed the ideas of [7] and Dots [5] is an online and near-optimal trajectory

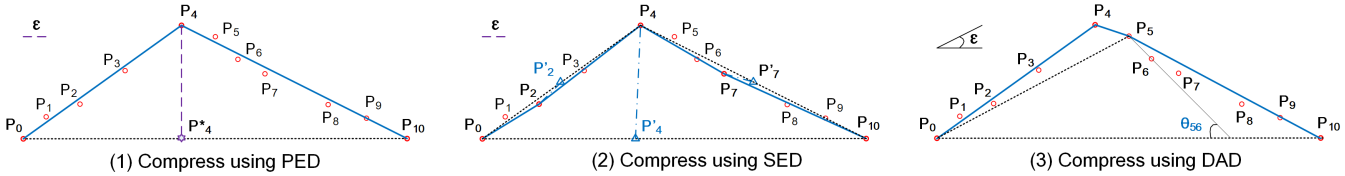

Figure 19: A trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ with 11 points is represented by (2) two and (3) four continuous line segments (solid blue), compressed by the Douglas–Peucker algorithm [9] with error metrics PED and SED, respectively.

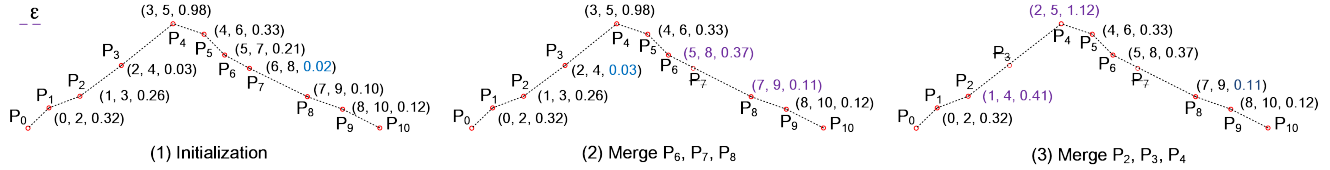

Figure 20: The trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_8]$ is compressed by the Theo-Pavlidis algorithm using PED to two line segments. The triple $(i, j, cost)$ is the *cost* of merging the line segments $\overline{P_iP_t}$ and $\overline{P_tP_j}$.

simplification algorithm that uses ISSED. Note that ISSED is a cumulative error that leads to varied effectiveness compared with SED.

Algorithms on the min-ϵ problem. Some work focuses on the minimal ϵ problem [6] that, given m, constructs an approximate curve consisting of at most m line segments with minimum error. SQUISH(λ) [24] and SQUISH-E(λ) [25] are such algorithms that compress a trajectory of length n into a trajectory of length at most n/λ. These methods lack the capability of compressing trajectories while ensuring that SED errors are within a user-specified bound [25].

Appendix: Examples

Example 1: Consider Figure 19, 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 piecewise line representations of trajectory $\ddot{\mathcal{T}}$, (3) $ped(P_4, \overrightarrow{P_0P_{10}}) = |\overrightarrow{P_4P_4^*}|$, 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) $sed(P_4, \overrightarrow{P_0P_{10}}) = |\overrightarrow{P_4P_4'}|$, $sed(P_2, \overrightarrow{P_0P_4}) = |\overrightarrow{P_2P_2'}|$ and $sed(P_7, \overrightarrow{P_4P_{10}}) = |\overrightarrow{P_7P_7'}|$, 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) $dad(\overrightarrow{P_5P_6}, \overrightarrow{P_0P_{10}}) = \theta_{56}$ is the DAD of line segment $\overrightarrow{P_5P_6}$ to $\overrightarrow{P_0P_{10}}$. □

Example 2: Consider the trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ shown in Figure 19. 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. □

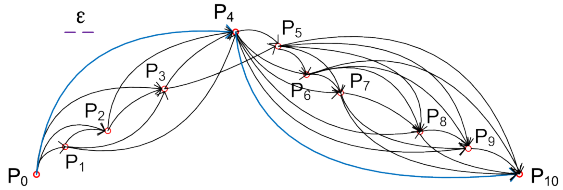

Figure 21: Example of reachability graph of trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ whose shortest path is (P_0, P_4, P_{10}).

Example 3: Figure 20 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) $\overrightarrow{P_2P_3}$ and $\overrightarrow{P_3P_4}$ are merged to $\overrightarrow{P_2P_4}$. The cost merging $\overrightarrow{P_2P_4}$ and $\overrightarrow{P_4P_5}$, and the cost of merging $\overrightarrow{P_1P_2}$ and $\overrightarrow{P_2P_4}$ are also updated. (4) At last, the algorithm outputs two line segments $\overrightarrow{P_0P_4}$ and $\overrightarrow{P_4P_{10}}$. □

Example 4: Figure 21 is an example of the Optimal algorithm using PED taking as input the trajectory $\ddot{\mathcal{T}}$ shown in Figure 19. 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 $\overrightarrow{P_4P_{10}}$. □

Example 5: Figure 22 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_3 has the minimal value, thus, it is removed from the list. The *mnprio* properties of P_2 and P_4 are updated to $max\{mnprio(pre(P_3)), prio(P_3)\} = max\{mnprio(P_2), prio(P_3)\} = max\{0, 0.39\epsilon\} = 0.39\epsilon$,

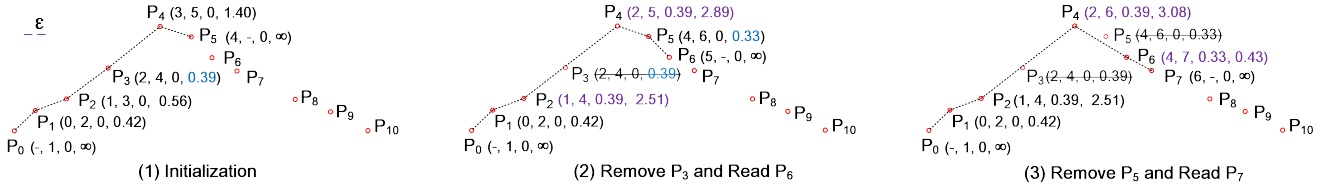

Figure 22: The trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ is compressed by the SQUISH-E algorithm using SED to five line segments. The size of Q is 6, and the data structure after point P is a tuple $(pre, suc, mmprio, prio)$.

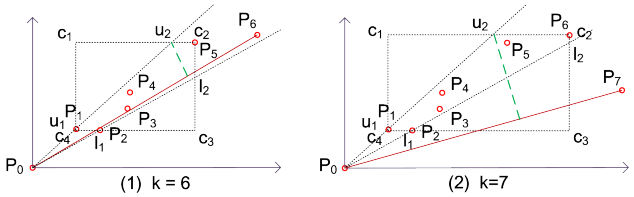

Figure 23: Examples for algorithm BQS.

and $max\{mnprio(P_4),\ prio(P_3)\} = 0.39\epsilon$, respectively. Furthermore, the *prio* property of P_4 is updated to $mnprio(suc(P_j)) + sed(suc(P_j), \overrightarrow{pre(P_j)suc(suc(P_j))}) = mnprio(P_4) + sed(P_4, \overrightarrow{P_2P_5}) = 0.39\epsilon + 2.50\epsilon = 2.89\epsilon$, and the *prio* property of P_2 is updated to $mnprio(P_2) + sed(P_2, \overrightarrow{P_1P_4}) = 0.39\epsilon + 2.12\epsilon = 2.51\epsilon$. Then, P_6 is read, and the information of P_5 is updated. (3) P_5 is removed and P_7 is read to the list. (4) Finally, the algorithm outputs 5 line segments $\overrightarrow{P_0P_2}, \overrightarrow{P_2P_4}, \overrightarrow{P_4P_7}, \overrightarrow{P_7P_9}$ and $\overrightarrow{P_9P_{10}}$. □

Example 6: Figure 23 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). □

Example 7: Figure 24 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 start 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 $\overrightarrow{P_4P_{10}}$. □

Example 8: Figure 25 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 border lines. 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 $\overrightarrow{P_4P_{10}}$. □

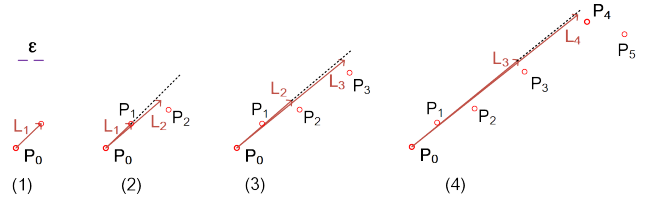

Figure 24: The trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ is compressed by the OPERB algorithm using PED to two line segments.

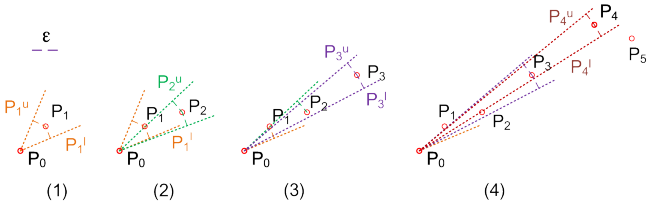

Figure 25: The trajectory $\ddot{\mathcal{T}}$ is compressed by the sector intersection algorithm using PED to two line segments.

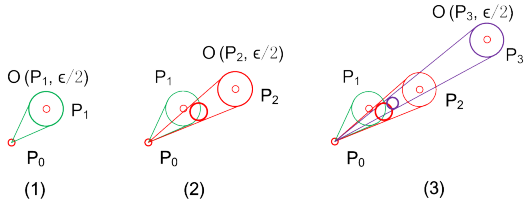

Figure 26: A running example of the CISED algorithm. The points and the oblique circular cones are projected on an x-y space.

Example 9: Figure 26 shows a running example of CISED ($\frac{\epsilon}{2}$) for compressing the trajectory $\ddot{\mathcal{T}}$ in Figure 19. For convenience, we project the points and the oblique circular cones on a x-y space. (1) After initialization, the CISED algorithm reads point P_1 and builds a narrow *oblique circular cone* $\mathcal{C}(P_0, \mathcal{O}(P_1, \epsilon/2))$, taking P_0 as its apex and $\mathcal{O}(P_1, \epsilon/2)$ 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 $\mathcal{C}(P_0, \mathcal{O}(P_2, \epsilon/2))$ (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

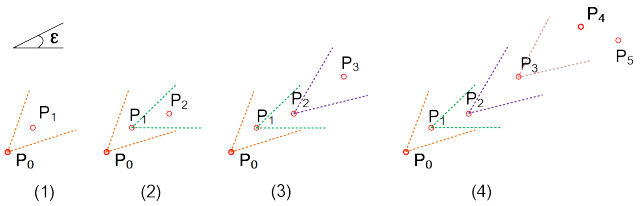

Figure 27: The trajectory $\dddot{\mathcal{T}}$ is compressed by the interval algorithm using DAD to two line segments.

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.,* $\{\overrightarrow{P_0P_2}, \overrightarrow{P_2P_4}, \overrightarrow{P_4P_7}, \overrightarrow{P_7P_{10}}\}$. □

Example 10: Figure 27 is a running example of the *interval* method taking as input the same trajectory $\dddot{\mathcal{T}}[P_0, \ldots, P_{10}]$. 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 subinterval*, and $\bigcap_{i=1}^{5} Range(\overrightarrow{P_{0+i-1}P_{0+i}}, \epsilon) = \phi$, $\overrightarrow{P_0P_4}$ is output and P_4 becomes the start point of the next section. At last, the algorithm outputs two continuous line segments $\overrightarrow{P_0P_4}$ and $\overrightarrow{P_4P_{10}}$. □