One-pass trajectory simplification using the synchronous Euclidean distance

Xuelian Lin, Jiahao Jiang, Shuai Ma, Yimeng Zuo & Chunming Hu

The VLDB Journal
The International Journal on Very Large Data Bases

ISSN 1066-8888
Volume 28
Number 6

The VLDB Journal (2019) 28:897-921
DOI 10.1007/s00778-019-00575-8

Springer

The VLDB Journal (2019) 28:897–921
https://doi.org/10.1007/s00778-019-00575-8

REGULAR PAPER

One-pass trajectory simplification using the synchronous Euclidean distance

Xuelian Lin[1] · Jiahao Jiang[1] · Shuai Ma[1] · Yimeng Zuo[1] · Chunming Hu[1]

Received: 9 January 2018 / Revised: 21 August 2019 / Accepted: 21 September 2019 / Published online: 4 October 2019

Abstract

Various mobile devices have been used to collect, store and transmit tremendous trajectory data, and it is known that raw trajectory data seriously wastes the storage, network bandwidth and computing resource. To attack this issue, one-pass line simplification (LS) algorithms have been developed, by compressing data points in a trajectory to a set of continuous line segments. However, these algorithms adopt the *perpendicular Euclidean distance*, and none of them uses the *synchronous Euclidean distance* (SED), and cannot support spatiotemporal queries. To do this, we develop two one-pass error bounded trajectory simplification algorithms (CISED-S and CISED-W) using SED, based on a novel spatiotemporal cone intersection technique. Using four real-life trajectory datasets, we experimentally show that our approaches are both efficient and effective. In terms of running time, algorithms CISED-S and CISED-W are on average 3 times faster than SQUISH-E (the fastest existing LS algorithm using SED). In terms of compression ratios, CISED-S is close to and CISED-W is on average 19.6% better than DPSED (the existing sub-optimal LS algorithm using SED and having the best compression ratios), and they are 21.1% and 42.4% better than SQUISH-E on average, respectively.

Keywords Trajectory simplification · Synchronous Euclidean distance · One-pass line simplification

1 Introduction

Various mobile devices, such as smart-phones, on-board diagnostics, personal navigation devices and wearable smart devices, use their sensors to collect massive trajectory data of moving objects at a certain sampling rate (*e.g.,* a data point every 5 s), which is then transmitted to cloud servers for various applications such as location based services and trajectory mining. Transmitting and storing raw trajectory data consumes too much network bandwidth and storage capacity [1,3,14–16,19,22–24,28,35]. These issues are commonly resolved or greatly alleviated by trajectory compression techniques [1,3,4,6,9,11,14–17,19,23,24,28], among which the piecewise line simplification technique is widely used [1,3,4,6,14–16,19,23], due to its distinct advantages: (a) simple and easy to implement, (b) no need of extra knowledge and suitable for freely moving objects and (c) bounded errors with good compression ratios [14,28].

Originally, line simplification (LS) algorithms adopt the *perpendicular Euclidean distance* (PED) as a metric to compute the errors. Suppose that a sub-trajectory $[P_s, \ldots, P_e]$ is represented by a line segment $\overrightarrow{P_sP_e}$ produced by an error bounded LS algorithm using PED. Then for any point $P \in \{P_s, \ldots, P_e\}$, its *perpendicular Euclidean distance* to the line segment $\overrightarrow{P_sP_e}$ is the shortest distance from P to $\overrightarrow{P_sP_e}$. Indeed, line segment $\overrightarrow{P_sP_e}$ represents all points that fall into an effective zone consisting of a rectangle and two half-circles, such that each point in the zone has a PED to $\overrightarrow{P_sP_e}$ not more than a predefined error bound ϵ, as shown in Fig. 1a. A typical issue of using PED is that the exact location of a point is hard to tell when the zone is large (we just know it is located in the zone). That leads to that the answer to a spatiotemporal query "*the position P of a moving object at time t* [1]" on

✉ Shuai Ma
mashuai@buaa.edu.cn

✉ Chunming Hu
hucm@buaa.edu.cn

Xuelian Lin
linxl@buaa.edu.cn

Jiahao Jiang
jiangjh@buaa.edu.cn

Yimeng Zuo
zuoym@buaa.edu.cn

1 Beijing Advanced Innovation Center for Big Data and Brain Computing (BDBC), Beihang University, Beijing, China

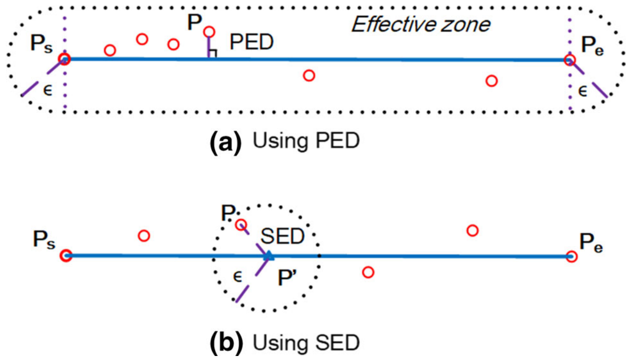

Fig. 1 A sub-trajectory $[P_s, \ldots, P_e]$ is simplified using PED and SED, respectively. A spatiotemporal query "*the position P of the moving object at time t*" on the simplified trajectory returns a point **a** in a large zone consisting of a rectangle and two half-circles, or **b** inside a small circle

the compressed trajectories is not bounded. That is, there is no way to find an approximate point P' of P at t such that their distance is bounded within ϵ. This shows that trajectory simplification using PED is not suitable for spatiotemporal queries that are necessary for location based services.

The *synchronous Euclidean distance* (SED) was then introduced for trajectory compression [1,4,19,23,29] to address the above issue for supporting bounded spatiotemporal queries. The use of SED highly depends on a notion named *synchronized point* [1,19] that could be computed conveniently. As shown in Fig. 1b, the *synchronized point* P' of a point P at time t *w.r.t.* a line segment $\overrightarrow{P_sP_e}$ is the expected position of the moving object on $\overrightarrow{P_sP_e}$ at time t *with the assumption that the object moves along a straight line from points P_s to P_e at a uniform speed* [1], *i.e.,* the average speed from points P_s to P_e. The SED of point P to line segment $\overrightarrow{P_sP_e}$ is the Euclidean distance between P and its *synchronized point P' w.r.t.* the line segment $\overrightarrow{P_sP_e}$. When an error bounded LS algorithm using SED is adopted, it requires that a point P is located within a circle with its synchronized point P' as its center and ϵ as its radius, as shown in Fig. 1b. Hence, the above spatiotemporal query over the trajectories compressed by SED can return the synchronized point P' as the approximate point of P at time t such that their distance is bounded within ϵ. Note that the SED of a point to a line segment is equal to or less than the PED of the point to the line segment as illustrated in Fig. 1b, and, hence, LS algorithms using SED typically generate more line segments.

The problem of finding the minimal number of line segments to represent the original polygonal lines *w.r.t.* an error bound ϵ is known as the "min-#" problem [2,13], and there exists an optimal LS algorithm, in terms of compression, using SED that runs in $O(n^3)$ [13] (originally designed for PED), where n is the number of the original points. Due to this high time complexity, sub-optimal LS algorithms using SED have been developed for trajectory compression, including batch algorithms (*e.g.,* Douglas–Peucker-based algorithm DPSED [19]) and online algorithms (*e.g.,* SQUISH-E [23]). However, these methods still have high time and/or space complexities, which hinders their utilities in resource-constrained devices.

Observe that one-pass LS algorithms using PED [7,14,36, 40,42] have been developed, and they are more efficient for resource-constrained devices. The key idea to achieve one-pass processing is by local distance checking for a single data point in constant time, *e.g.,* the *sector intersection* mechanism used in [7,36,40,42] and the *fitting function* approach used in our preview work [14]. Unfortunately, these techniques are designed specifically for PED, and work in a 2D space, not in a spatiotemporal 3D space that SED needs. Hence, they can hardly be applied for SED.

Indeed, it is even more challenging to design one-pass LS algorithms using SED than using PED. As SED introduces the temporal information besides the spatial information, a new local distance checking method in a spatiotemporal 3D space is needed. To our knowledge, no one-pass LS algorithms using SED have been developed in the community yet.

Contributions. To this end, we propose two fast one-pass error bounded LS algorithms using SED for compressing trajectories with good compression ratios.

(1) We first substantially extend the one-pass local distance checking approach from a 2D space to a spatiotemporal 3D space and develop a novel local synchronous distance checking approach, *i.e.,* spatiotemporal Cone Intersection using the Synchronous Euclidean Distance (CISED), such that each data point in a trajectory is checked in $O(1)$ time during the entire process of trajectory simplification. It is the first local distance checking method for trajectory compression using SED, and it is also the key to develop one-pass trajectory simplification algorithms using SED.

(2) We develop a method that finds the approximate common intersection of n circles in a linear time and a constant space, whose key idea is to approximate circles by a special class of regular polygons and compute their intersection with a fast regular polygon intersection algorithm. This also has a potential usage as a basic approximate function though we develop the method for the local synchronous distance checking.

(3) We design two one-pass trajectory simplification algorithms CISED-S and CISED-W, achieving $O(n)$ time complexity and $O(1)$ space complexity, based on our local synchronous distance checking technique. Algorithm CISED-S belongs to strong simplification that only has original points in its outputs, while algorithm CISED-W belongs to weak simplification that allows interpolated data points in its outputs.

(4) Using four real-life trajectory datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), we finally conduct an exten-

sive experimental study, by comparing our methods CISED-S and CISED-W with the compression optimal LS algorithm using SED(C-Optimal in short) [13], batch algorithm DPSED [19] (the existing sub-optimal LS algorithm using SED having the best compression ratios) and online algorithm SQUISH-E [23] (the fastest existing LS algorithm using SED). For running time, algorithms CISED-S and CISED-W are on average 15.0, 3.2 and 14345.0 times faster than DPSED, SQUISH-E and C-Optimal on the test datasets, respectively. For compression ratios, algorithm CISED-S is better than SQUISH-E and close to DPSED. The output sizes of CISED-S are on average 74.4%, 110.4% and 137.9% of SQUISH-E, DPSED and C-Optimal on the test datasets, respectively. Moreover, algorithm CISED-W is on average 54.9% and 81.6% better than SQUISH-E and DPSED on the test datasets, respectively.

It is worth pointing out that trajectory data is collected by mobile devices from GPS sensors, and these devices have range errors, which leads to data quality issues of trajectory data [27,44]. However, the problem is beyond the scope of this study, and we focus on lossy simplification of trajectory data only.

Organization. The remainder of the article is organized as follows. Section 2 introduces the basic concepts and techniques. Section 3 presents our local synchronous distance checking method. Section 4 presents our one-pass trajectory simplification algorithms. Section 5 reports the experimental results, followed by related work in Sect. 6 and conclusion in Sect. 7. All proofs are provided in the Appendix.

2 Preliminaries

In this section, we first introduce basic concepts for piecewise line-based trajectory compression. We then describe the compression optimal LS algorithm and the *sector intersection* mechanism, and show how this mechanism can be used to speed up the LS algorithms using PED and why it cannot work with SED. Finally, we illustrate a convex polygon intersection algorithm, which serves as one of the fundamental components of our local synchronous distance checking method. For convenience, notations used are summarized in Table 1.

2.1 Basic notations

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

Trajectories ($\ddot{\mathcal{T}}$). A trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ is a sequence of data points in a monotonically increasing order of their associated time values (*i.e.,* $P_i.t < P_j.t$ for any $0 \le i < j \le n$). Intuitively, a trajectory is the path (or track) that a moving object follows through space as a function of time [20].

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

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

Piecewise line representation ($\overline{\mathcal{T}}$). A piecewise line representation $\overline{\mathcal{T}}[\mathcal{L}_0, \ldots, \mathcal{L}_m]$ $(0 < m \le n)$ of a trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ is a sequence of continuous directed line segments $\mathcal{L}_i = \overrightarrow{P_{s_i} P_{e_i}}$ $(i \in [0, m])$ of $\ddot{\mathcal{T}}$ such that $\mathcal{L}_0.P_{s_0} = P_0$, $\mathcal{L}_m.P_{e_m} = P_n$ and $\mathcal{L}_i.P_{e_i} = \mathcal{L}_{i+1}.P_{s_{i+1}}$ for all $i \in [0, m-1]$. Note that each directed line segment in $\overline{\mathcal{T}}$ essentially represents a continuous sequence of data points in $\ddot{\mathcal{T}}$.

Perpendicular Euclidean Distance (PED). Given a data point P and a directed line segment $\mathcal{L} = \overrightarrow{P_s P_e}$, the perpendicular Euclidean distance (or simply perpendicular distance) $ped(P, \mathcal{L})$ of point P to line segment $\mathcal{L}$ is $\min\{|\overrightarrow{PQ}|\}$ for any point Q on $\overrightarrow{P_s P_e}$.

Synchronized points [19]. Given a sub-trajectory $\ddot{\mathcal{T}}_s[P_s, \ldots, P_e]$, the synchronized point P' of a data point $P \in \ddot{\mathcal{T}}_s$, *w.r.t.* line segment $\overrightarrow{P_s P_e}$ is defined as follows: (1) $P'.x = P_s.x + w \cdot (P_e.x - P_s.x)$, (2) $P'.y = P_s.y + w \cdot (P_e.y - P_s.y)$ and (3) $P'.t = P.t$, where $w = \frac{P.t - P_s.t}{P_e.t - P_s.t}$.

Synchronized points are essentially virtual points with the assumption that an object moved along a straight line from P_s to P_e with a uniform speed, *i.e.,* the average speed $\frac{|\overrightarrow{P_s P_e}|}{P_e.t - P_s.t}$ between points P_s and P_e [1]. Then, the *synchronized point* P' of a point P *w.r.t.* the line segment $\overrightarrow{P_s P_e}$ is the expected position of the moving object on $\overrightarrow{P_s P_e}$ at time $P.t$, obtained by a linear interpolation [1]. More specifically, a synchronized point P'_i of P_i $(s \le i < e)$ is a point on $\overrightarrow{P_s P_e}$ satisfying $|\overrightarrow{P_s P'_i}| = \frac{P_i.t - P_s.t}{P_e.t - P_s.t} \cdot |\overrightarrow{P_s P_e}|$, which means that the object moves from P_s to P_e at an average speed $\frac{|\overrightarrow{P_s P_e}|}{P_e.t - P_s.t}$, and its position at time $P_i.t$ is the point P'_i on $\overrightarrow{P_s P_e}$ having a distance of $\frac{P_i.t - P_s.t}{P_e.t - P_s.t} \cdot |\overrightarrow{P_s P_e}|$ to P_s [1,4,19,41].

Synchronous Euclidean Distance (SED) [19]. Given a data point P and a directed line segment $\mathcal{L} = \overrightarrow{P_s P_e}$, the synchronous Euclidean distance (or simply synchronous distance) $sed(P, \mathcal{L})$ of P to $\mathcal{L}$ is $|\overrightarrow{PP'}|$ that is the Euclidean distance from P to its synchronized point P' *w.r.t.* $\mathcal{L}$.

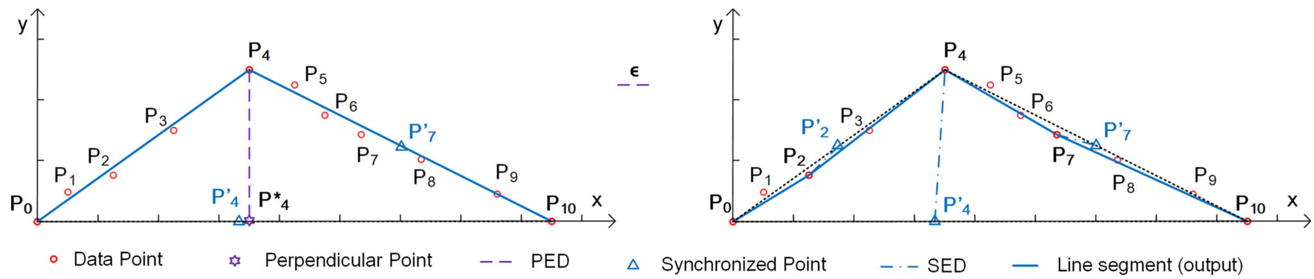

Fig. 2 A trajectory $\dddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ with eleven points is represented by two (left) and four (right) continuous line segments (solid blue), compressed by the Douglas–Peucker algorithm [6] using PED and SED, respectively. The Douglas–Peucker algorithm firstly creates line segment $\overrightarrow{P_0P_{10}}$; then, it calculates the distance of each point in the trajectory to $\overrightarrow{P_0P_{10}}$. It finds that point P_4 has the maximum distance to $\overrightarrow{P_0P_{10}}$, and is greater than the user-defined threshold ϵ. Then, it goes to compress sub-trajectories $[P_0, \ldots, P_4]$ and $[P_4, \ldots, P_{10}]$, separately

Table 1 Summary of notations

Notations	Semantics
P	A data point
$\dddot{\mathcal{T}}$	A trajectory $\dddot{\mathcal{T}}$ is a sequence of data points
$\overline{\mathcal{T}}$	A piecewise line representation of a trajectory $\dddot{\mathcal{T}}$
$\mathcal{L}$	A directed line segment
$\overrightarrow{P_sP_e}$	A directed line segment with the start point P_s and the end point P_e
$\lvert\mathcal{L}\rvert$	The length of $\mathcal{L}$ in the x-y 2D space
$ped(P, \mathcal{L})$	The perpendicular Euclidean distance of point P to line segment $\mathcal{L}$
$sed(P, \mathcal{L})$	The synchronous Euclidean distance of point P to line segment $\mathcal{L}$
ϵ	The error bound
$\mathcal{S}$	A sector
$\overrightarrow{A} \times \overrightarrow{B}$	The cross product of (vectors) $\overrightarrow{A}$ and $\overrightarrow{B}$
$\mathcal{H}(\mathcal{L})$	The open half-plane to the left of $\mathcal{L}$
$\mathcal{R}$	A convex polygon
$\mathcal{R}^*$	The intersection of convex polygons
m	The maximum number of edges of a polygon
E^j	A group of edges labeled with j
$g(e)$	The label of an edge e of polygons
$\mathcal{O}$	A synchronous circle
$\mathcal{C}$	A spatiotemporal cone
$\mathcal{O}^c$	A cone projection circle
$\sqcap$	Intersection of geometries
G	The reachability graph of a trajectory

The use of PED brings better compression ratios, but the temporal information of data points is not available [19], which makes PED not suitable for spatiotemporal queries. In contrast, SED takes both spatial and temporal information of data points into consideration [19]. Hence, SED is more suitable for spatiotemporal queries.

We illustrate these notations with examples.

Example 1 Consider Fig. 2, in which
(1) $\dddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ is a trajectory having 11 data points,
(2) the set of two continuous line segments $\{\overrightarrow{P_0P_4}, \overrightarrow{P_4P_{10}}\}$ (Left) and the set of four continuous line segments $\{\overrightarrow{P_0P_2}, \overrightarrow{P_2P_4}, \overrightarrow{P_4P_7}, \overrightarrow{P_7P_{10}}\}$ (Right) are two piecewise line representations of trajectory $\dddot{\mathcal{T}}$,
(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) P_4' is the synchronized point of P_4 *w.r.t.* line segment $\overrightarrow{P_0P_{10}}$, satisfying $\frac{|\overrightarrow{P_0P_4'}|}{|\overrightarrow{P_0P_{10}}|} = \frac{P_4.t - P_0.t}{P_{10}.t - P_0.t} = \frac{4-0}{10-0} = \frac{2}{5}$, and

(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. □

Error bounded algorithms. Given a trajectory $\dddot{\mathcal{T}}$ and its compression algorithm $\mathcal{A}$ using SED (respectively, PED) that produces another trajectory $\dddot{\mathcal{T}}'$, we say that algorithm $\mathcal{A}$ is error bounded by ϵ if for each point P in $\dddot{\mathcal{T}}$, there exist points P_j and P_{j+1} in $\dddot{\mathcal{T}}'$ such that $P_j.t \le P.t \le P_{j+1}.t$ and $sed(P, \mathcal{L}(P_j, P_{j+1})) \le \epsilon$ (respectively, $ped(P, \mathcal{L}(P_j, P_{j+1})) \le \epsilon$).

2.2 The compression optimal LS algorithm

Given a trajectory $\dddot{\mathcal{T}}[P_0, \ldots, P_n]$ and an error bound ϵ, the compression optimal trajectory simplification problem, as formulated by Imai and Iri in [13], can be solved in two steps: (1) construct a reachability graph G of $\dddot{\mathcal{T}}$, and (2) search a shortest path from P_0 to P_n in G.

The reachability graph of a trajectory $\dddot{\mathcal{T}}[P_0, \ldots, P_n]$ *w.r.t.* a bound ϵ is an unweighted graph $G(V, E)$, where (1) $V = \{P_0, \ldots, P_n\}$, and (2) for any nodes P_s and $P_{s+k} \in V$ ($s \ge 0, k > 0, s + k \le n$), edge $(P_s, P_{s+k}) \in E$ if and only if the distance of each point P_{s+i} ($i \in [0, k]$) to line segment $\overrightarrow{P_sP_{s+k}}$ is not greater than ϵ.

Observe that in the reachability graph G, (1) a path from nodes P_0 to P_n is a representation of trajectory $\dddot{\mathcal{T}}$. The path also reveals the subset of points of $\dddot{\mathcal{T}}$ used in the approximate trajectory, (2) the path length corresponds to the number of line segments in the approximate trajectory, and (3) a shortest path is a compression optimal representation of trajectory $\dddot{\mathcal{T}}$.

Constructing the reachability graph G needs to check for all pairs of points P_s and P_{s+k} whether the distances of all points P_{s+i} ($0 < i < k$) to the line segment $\overrightarrow{P_sP_{s+k}}$ are less than ϵ. There are $O(n^2)$ pairs of points in the trajectory and checking the error of all points P_{s+i} to a line segment $\overrightarrow{P_sP_{s+k}}$ takes $O(n)$ time. Thus, the construction step takes $O(n^3)$ time. Finding shortest paths on unweighted graphs can be done in linear time. Hence, the brute-force algorithm takes $O(n^3)$ time in total.

Though the brute-force algorithm was initially developed using PED, it can be used for SED. As pointed out in [2], the construction of the reachability graph G using PED can be implemented in $O(n^2)$ time using the *sector intersection* mechanism (see Sect. 2.3). However, the *sector intersection* mechanism cannot work with SED. Hence, the construction of the reachability graph G using SED remains in $O(n^3)$ time, and the brute-force algorithm using SED remains in $O(n^3)$ time.

2.3 Sector intersection-based algorithms using PED

The sector intersection (SI) algorithm [36,40] was developed for graphic and pattern recognition in the late 1970s, for the approximation of arbitrary planar curves by linear segments or finding a polygonal approximation of a set of input data points in a 2D Cartesian coordinate system. The Sleeve algorithm [42] in the cartographic discipline essentially applies the same idea as the SI algorithm. Further, [7] optimized algorithm SI by considering the distance between a potential end point and the initial point of a line segment. It is worth pointing out that all these SI-based algorithms use the perpendicular Euclidean distance.

Given a sequence of data points $[P_s, P_{s+1}, \ldots, P_{s+k}]$ and an error bound ϵ, the SI-based algorithms process the input points one by one in order and produce a simplified polyline. Instead of using the distance threshold ϵ directly, the SI-based algorithms convert the distance tolerance into a variable angle tolerance for testing the successive data points.

For the start data point P_s, any point P_{s+i} and $|\overrightarrow{P_sP_{s+i}}| > \epsilon$ ($i \in [1, k]$), there are two directed lines $\overrightarrow{P_sP^u_{s+i}}$ and $\overrightarrow{P_sP^l_{s+i}}$ such that $ped(P_{s+i}, \overrightarrow{P_sP^u_{s+i}}) = ped(P_{s+i}, \overrightarrow{P_sP^l_{s+i}}) = \epsilon$ and either ($\overrightarrow{P_sP^l_{s+i}}.\theta < \overrightarrow{P_sP^u_{s+i}}.\theta$ *and* $\overrightarrow{P_sP^u_{s+i}}.\theta - \overrightarrow{P_sP^l_{s+i}}.\theta < \pi$) or ($\overrightarrow{P_sP^l_{s+i}}.\theta > \overrightarrow{P_sP^u_{s+i}}.\theta$ *and* $\overrightarrow{P_sP^u_{s+i}}.\theta - \overrightarrow{P_sP^l_{s+i}}.\theta < -\pi$). Indeed, they form a *sector* $\mathcal{S}(P_s, P_{s+i}, \epsilon)$ that takes P_s as the center point and $\overrightarrow{P_sP^u_{s+i}}$ and $\overrightarrow{P_sP^l_{s+i}}$ as the border lines. Then, 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 $\overrightarrow{P_sQ}$ is not greater than the error bound ϵ if and only if the k sectors $\mathcal{S}(P_s, P_{s+i}, \epsilon)$ ($i \in [1, k]$) share common data points other than P_s, *i.e.*, $\sqcap_{i=1}^{k}\mathcal{S}(P_s, P_{s+i}, \epsilon) \ne \{P_s\}$ [36,40,42].

The point Q may not belong to $\{P_s, P_{s+1}, \ldots, P_{s+k}\}$. However, if P_{s+i} ($1 \le i \le k$) is chosen as Q, then for any data point P_{s+j} ($j \in [1, ...i]$), its perpendicular Euclidean distance to line segment $\overrightarrow{P_sP_{s+i}}$ is not greater than the error bound ϵ if and only if $\sqcap_{j=1}^{i}\mathcal{S}(P_s, P_{s+j}, \epsilon/2) \ne \{P_s\}$, as pointed out in [42].

That is, *these* SI *-based algorithms can be easily adopted for trajectory compression using* PED*although they have been overlooked by existing trajectory simplification studies*. The SI-based algorithms run in $O(n)$ time and $O(1)$ space and are one-pass algorithms.

We next illustrate how the SI-based algorithms can be used for trajectory compression with an example.

Example 2 Consider Fig. 3. An SI-based simplification algorithm takes as input a trajectory $\dddot{\mathcal{T}}[P_0, \ldots, P_{10}]$, and returns a simplified polyline consisting of two line segments $\overrightarrow{P_0P_4}$ and $\overrightarrow{P_4P_{10}}$. Initially, point P_0 is the start point.

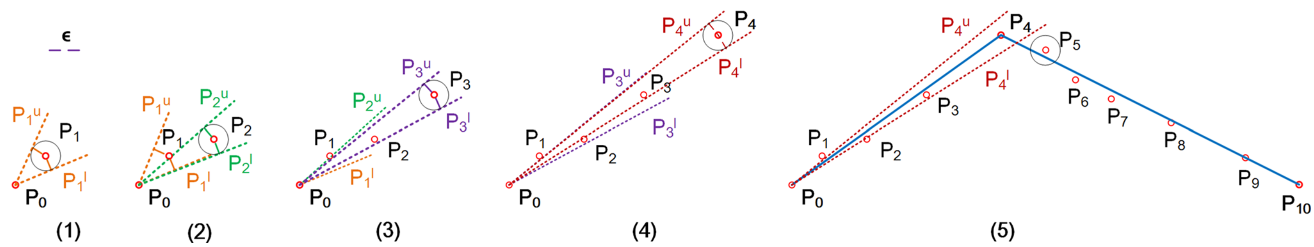

Fig. 3 Trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ in Fig. 2 is compressed into two line segments by the sector intersection algorithm [36,40]. Each circle in the figure has a radius of $\epsilon/2$, which is used to define the sector

(1) Point P_1 is firstly read, and the sector $\mathcal{S}(P_0, P_1, \epsilon/2)$ of P_1 is created as shown in Fig. 3(1).

(2) Then, P_2 is read, and the sector $\mathcal{S}(P_0, P_2, \epsilon/2)$ is created for P_2. The intersection of sectors $\mathcal{S}(P_0, P_1, \epsilon/2)$ and $\mathcal{S}(P_0, P_2, \epsilon/2)$ contains data points other than P_0, which has an up border line $\overrightarrow{P_0 P_2^u}$ and a low border line $\overrightarrow{P_0 P_1^l}$, as shown in Fig. 3(2).

Similarly, points P_3 and P_4 are processed, as shown in Fig. 3(3), (4), respectively.

(3) When point P_5 is read, line segment $\overrightarrow{P_0 P_4}$ is produced, and point P_4 becomes the start point, as $\sqcap_{i=1}^{4} \mathcal{S}(P_0, P_{s+i}, \epsilon/2) \neq \{P_0\}$ and $\sqcap_{i=1}^{5} \mathcal{S}(P_0, P_{s+i}, \epsilon/2) = \{P_0\}$ as shown in Fig. 3(5).

(4) Points $P_5, \ldots, P_{10}$ are processed similarly one by one in order, and finally, the algorithm outputs another line segment $\overrightarrow{P_4 P_{10}}$ as shown in Fig. 3(5). □

However, if we use SED instead of PED, then "the k sectors $\mathcal{S}(P_s, P_{s+i}, \epsilon)$ $(i \in [1, k])$ sharing common data points other than P_s" cannot ensure "there exists a data point Q such that for any data point P_{s+i} $(i \in [1, \ldots k])$, its synchronous Euclidean distance to directed line $\overline{P_s Q}$ is not greater than the error bound ϵ." Hence, the SI mechanism is PED specific, and not applicable for SED.

2.4 Intersection computation of convex polygons

We also employ and revise a convex polygon intersection algorithm developed in [25], whose basic idea is straightforward. Assume *w.l.o.g.* that the edges of polygons $\mathcal{R}_1$ and $\mathcal{R}_2$ are oriented counterclockwise, and $\vec{A} = (P_{s_A}, P_{e_A})$ and $\vec{B} = (P_{s_B}, P_{e_B})$ are two (directed) edges on $\mathcal{R}_2$ and $\mathcal{R}_1$, respectively.

The algorithm has $\vec{A}$ and $\vec{B}$ "chasing" one another, *i.e.*, moves $\vec{A}$ on $\mathcal{R}_2$ and $\vec{B}$ on $\mathcal{R}_1$ counterclockwise step by step under certain rules, so that they meet at every crossing of $\mathcal{R}_1$ and $\mathcal{R}_2$. The rules, called *advance rules*, are carefully designed depending on geometric conditions of $\vec{A}$ and $\vec{B}$. Let $\vec{A} \times \vec{B}$ be the cross product of (vectors) $\vec{A}$ and $\vec{B}$, and

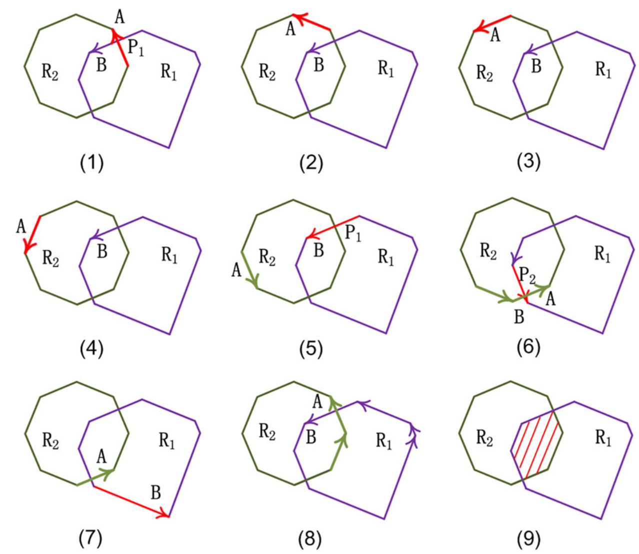

Fig. 4 A running example of convex polygon intersection

$\mathcal{H}(\vec{A})$ be the open half-plane to the left of $\vec{A}$, the rules are as follows:

Rule (1): If $\vec{A} \times \vec{B} < 0$ and $P_{e_A} \notin \mathcal{H}(\vec{B})$, or $\vec{A} \times \vec{B} \geq 0$ and $P_{e_B} \in \mathcal{H}(\vec{A})$, then $\vec{A}$ is advanced a step.

For example, in Fig. 4(1), (2), $\vec{A}$ moves forward a step as $\vec{A} \times \vec{B} > 0$ and $P_{e_B} \in \mathcal{H}(\vec{A})$.

Rule (2): If $\vec{A} \times \vec{B} \geq 0$ and $P_{e_B} \notin \mathcal{H}(\vec{A})$, or $\vec{A} \times \vec{B} < 0$ and $P_{e_A} \in \mathcal{H}(\vec{B})$, then $\vec{B}$ is advanced a step.

For example, in Fig. 4(6), (7), $\vec{B}$ moves forward a step as $\vec{A} \times \vec{B} < 0$ and $P_{e_A} \in \mathcal{H}(\vec{B})$.

Algorithm CPolyInter. The complete algorithm is shown in Fig. 5. Given two convex polygons $\mathcal{R}_1$ and $\mathcal{R}_2$, algorithm CPolyInter first arbitrarily sets directed edge $\vec{A}$ on $\mathcal{R}_2$ and directed edge $\vec{B}$ on $\mathcal{R}_1$, respectively (line 1). It then checks the intersection of edges $\vec{A}$ and $\vec{B}$. If $\vec{A}$ intersects $\vec{B}$ (line 3), then the algorithm checks for some special termination conditions (*e.g.*, if $\vec{A}$ and $\vec{B}$ are overlapped and, at the same time, polygons $\mathcal{R}_1$ and $\mathcal{R}_2$ are on the opposite sides of the overlapped edges, then the process is terminated) (line 4), and records the inner edge, which is a boundary segment of

Algorithm CPolyInter $(\mathcal{R}_1, \mathcal{R}_2)$
1. **set** $\vec{A}$ and $\vec{B}$ arbitrarily on $\mathcal{R}_2$ and $\mathcal{R}_1$, respectively;
2. **repeat**
3. **if** $\vec{A} \sqcap \vec{B} \neq \emptyset$ **then**
4. Check for termination;
5. Update an inside flag;
6. **if** $(\vec{A} \times \vec{B} < 0$ **and** $P_{e_A} \notin \mathcal{H}(\vec{B}))$ **or**
7. $(\vec{A} \times \vec{B} \geq 0$ **and** $P_{e_B} \in \mathcal{H}(\vec{A}))$ **then**
8. advance $\vec{A}$ one step;
9. **elseif** $(\vec{A} \times \vec{B} \geq 0$ **and** $P_{e_B} \notin \mathcal{H}(\vec{A}))$ **or**
10. $(\vec{A} \times \vec{B} < 0$ **and** $P_{e_A} \in \mathcal{H}(\vec{B}))$ **then**
11. advance $\vec{B}$ one step;
12. **until** both $\vec{A}$ and $\vec{B}$ cycle their polygons
13. **handle** $\mathcal{R}_1 \subset \mathcal{R}_2$ and $\mathcal{R}_2 \subset \mathcal{R}_1$ and $\mathcal{R}_1 \sqcap \mathcal{R}_2 = \emptyset$ cases;
14. **return** $\mathcal{R}_1 \sqcap \mathcal{R}_2$.

Fig. 5 Algorithm for convex polygon intersection [25]

the intersection polygon (line 5). After that, the algorithm moves on $\vec{A}$ or $\vec{B}$ one step under the advance rules (lines 6–11). The above processes repeated, until both $\vec{A}$ and $\vec{B}$ completely cycle their polygons (line 12). Next, the algorithm handles three special cases of the polygons $\mathcal{R}_1$ and $\mathcal{R}_2$, *i.e.,* $\mathcal{R}_1$ is inside of $\mathcal{R}_2$, $\mathcal{R}_2$ is inside of $\mathcal{R}_1$, and $\mathcal{R}_1 \sqcap \mathcal{R}_2 = \emptyset$ (line 13). At last, it returns the intersection polygon (line 14).

Example 3 Figure 4 shows a running example of the convex polygon intersection algorithm CPolyInter.

(1) Initially, directed edges $\vec{A}$ and $\vec{B}$ are on polygons $\mathcal{R}_2$ and $\mathcal{R}_1$, respectively, such that $\vec{A} \sqcap \vec{B} = \{P_1\}$, *i.e.,* $\vec{A}$ and $\vec{B}$ intersect on point P_1, as shown in Fig. 4(1).

(2) Then, because $\vec{A} \times \vec{B} > 0$ and $P_{e_B} \in \mathcal{H}(\vec{A})$, by the advance rule (1), edge $\vec{A}$ moves on a step and makes $\vec{A} \sqcap \vec{B} = \emptyset$ as shown in Fig. 4(2). After 7 steps of moving of edge $\vec{A}$ or $\vec{B}$, each by an advance rule, $\vec{A}$ and $\vec{B}$ intersect on P_2, as shown in Fig. 4(6).

(3) Next, because $\vec{A} \times \vec{B} < 0$ and $P_{e_A} \in \mathcal{H}(\vec{B})$, by the advance rule (2), edge $\vec{B}$ moves on a step, and makes $\vec{A} \sqcap \vec{B} = \emptyset$, as shown in Fig. 4(7).

(4) After 6 steps of moving of edge $\vec{B}$ or $\vec{A}$ one by one, both edges $\vec{A}$ and $\vec{B}$ have finished their cycles as shown in Fig. 4(8).

(5) The algorithm finally returns the intersection polygon as shown in Fig. 4(9). □

Algorithm CPolyInter has a time complexity of $O(|\mathcal{R}_1| + |\mathcal{R}_2|)$, where $|\mathcal{R}|$ is the number of edges of polygon $\mathcal{R}$. It is also worth pointing out that $|\mathcal{R}_1 \sqcap \mathcal{R}_2| \leq (|\mathcal{R}_1| + |\mathcal{R}_2|)$.

3 Local synchronous distance checking

In this section, we develop a local synchronous distance checking approach, laying down the key for the one-pass trajectory simplification algorithms using SED (Sect. 4), such that each point in a trajectory is checked only once in $O(1)$ time during the entire process of trajectory simplification. The local synchronous distance checking method is based on a new concept of *spatiotemporal cones* that converts the SED distance tolerance into the intersection of spatiotemporal cones for testing the successive data points. More specifically, we first substantially extend the *sectors* in Sect. 2.3 from a 2D space to a spatiotemporal 3D space, which leads to *spatiotemporal cones*. Then, we prove that the SED distance checking can be achieved by the intersection of spatiotemporal cones. Finally, we simplify the spatiotemporal cone intersection into the circle intersection and approximate circles with a special class of (fixed rotation and edge number) polygons.

We consider a sub-trajectory $\dddot{\mathcal{T}}_s[P_s, \ldots, P_{s+k}]$, an error bound ϵ, and a 3D Cartesian coordinate system whose origin, x-axis, y-axis and t-axis are P_s, longitude, latitude and time, respectively.

3.1 Spatiotemporal cone intersection

We first present the *spatiotemporal cone intersection* method in a 3D Cartesian coordinate system.

Synchronous circles $(\mathcal{O})$. The synchronous circle of a data point P_{s+i} $(1 \leq i \leq k)$ in $\dddot{\mathcal{T}}_s$ *w.r.t.* an error bound ϵ, denoted as $\mathcal{O}(P_{s+i}, \epsilon)$, or $\mathcal{O}_{s+i}$ in short, is a circle on the plane $P.t - P_{s+i}.t = 0$ such that P_{s+i} is its center and ϵ is its radius.

Spatiotemporal cones $(\mathcal{C})$. Given a start point P_s of $\dddot{\mathcal{T}}_s$ and an error bound ϵ, the spatiotemporal cone (or simply *cone*) of a data point P_{s+i} $(1 \leq i \leq k)$ in $\dddot{\mathcal{T}}_s$ *w.r.t.* P_s and 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 $\mathcal{O}(P_{s+i}, \epsilon)$ of point P_{s+i} is its base.

Example 4 (1) Figure 6 shows two synchronous circles, $\mathcal{O}(P_{s+i}, \epsilon)$ of point P_{s+i} and $\mathcal{O}(P_{s+k}, \epsilon)$ of point P_{s+k}. It is easy to see that for any point in the area of a circle $\mathcal{O}(P_{s+i}, \epsilon)$, its distance to P_{s+i} is not greater than ϵ.

(2) Figure 6 also illustrates two example spatiotemporal cones: $\mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon))$ (purple) and $\mathcal{C}(P_s, \mathcal{O}(P_{s+k}, \epsilon))$ (red), with the same apex P_s and error bound ϵ. □

Indeed, the SED distance tolerance can be checked by finding the common intersection of spatiotemporal cones, as shown below.

Proposition 1 *Given a sub-trajectory* $[P_s, \ldots, P_{s+k}]$ *and an error bound* ϵ, *there exists a point* Q *such that* $Q.t = P_{s+k}.t$ *and* $sed(P_{s+i}, \overrightarrow{P_s Q}) \leq \epsilon$ *for each* $i \in [1, k]$ *if and only if* $\sqcap_{i=1}^{k} \mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon)) \neq \{P_s\}$.

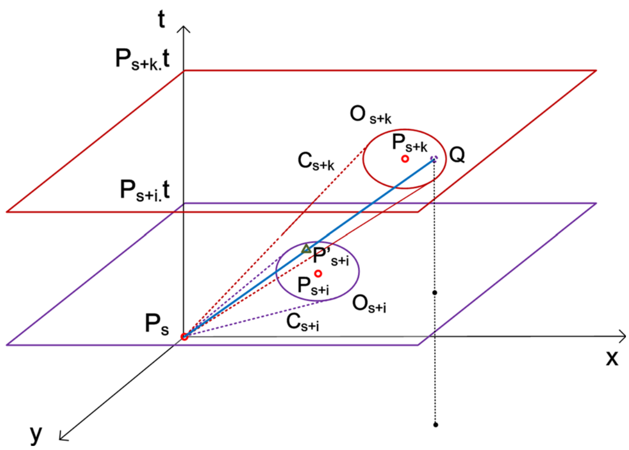

Fig. 6 Examples of spatiotemporal cones in a 3D Cartesian coordinate system, where (1) P_s, P_{s+i} and P_{s+k} are three points, (2) $\mathcal{O}_{s+i}$ and $\mathcal{O}_{s+k}$ are two synchronous circles, (3) $\mathcal{C}_{s+i}$ and $\mathcal{C}_{s+k}$ are two spatiotemporal cones, (4) Q is a point in synchronous circle $\mathcal{O}_{s+k}$, and (5) P'_{s+i} is the intersection point of line $\overrightarrow{P_sQ}$ and synchronous circle $\mathcal{O}_{s+i}$

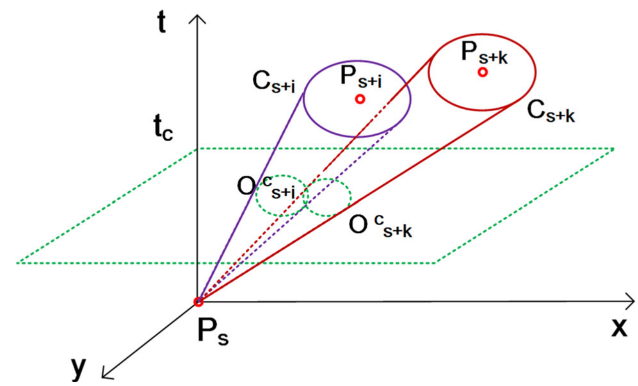

Fig. 7 Cone projection circles

By Proposition 1, we now have a spatiotemporal cone intersection method in a 3D Cartesian coordinate system, which significantly extends the sector intersection method [36,40,42] from a 2D space to a spatiotemporal 3D space.

3.2 Circle intersection

For spatiotemporal cones with the same apex P_s, we then show that 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.

Cone projection circles. The projection of a cone $\mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon))$ on a plane $P.t - t_c = 0$ ($t_c > P_s.t$) is a circle $\mathcal{O}^c(P^c_{s+i}, r^c_{s+i})$, or $\mathcal{O}^c{}_{s+i}$ in short, such that (1) $P^c_{s+i}.x = P_s.x + w \cdot (P_{s+i}.x - P_s.x)$, (2) $P^c_{s+i}.y = P_s.y + w \cdot (P_{s+i}.y - P_s.y)$, (3) $P^c_{s+i}.t = t_c$, and (4) $r^c_{s+i} = w \cdot \epsilon$, where $w = \frac{t_c - P_s.t}{P_{s+i}.t - P_s.t}$.

Recall that the base of a cone $\mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon))$ is a circle on plane $P.t - P_{s+i}.t = 0$, and plane $P.t - t_c = 0$ is parallel to plane $P.t - P_{s+i}.t = 0$. These ensure that the projection of a cone on plane $P.t - t_c = 0$ is a circle.

Example 5 In Fig. 7, the green dashed circles $\mathcal{O}^c(P^c_{s+i}, r^c_{s+i})$ and $\mathcal{O}^c(P^c_{s+k}, r^c_{s+k})$ on plane "$P.t - t_c = 0$" are the projection circles of cones $\mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon))$ and $\mathcal{C}(P_s, \mathcal{O}(P_{s+k}, \epsilon))$ on the plane. □

Proposition 2 *Given a sub-trajectory $[P_s, \ldots, P_{s+k}]$, an error bound ϵ, and any $t_c > P_s.t$, there exists a point Q such that $Q.t = P_{s+k}.t$ and $sed(P_{s+i}, \overrightarrow{P_sQ}) \le \epsilon$ for all points P_{s+i} ($i \in [1, k]$) if and only if $\bigcap_{i=1}^{k} \mathcal{O}^c(P^c_{s+i}, r^c_{s+i}) \neq \emptyset$.*

Proposition 2 tells us that the intersection checking of spatiotemporal cones can be reduced to simply check the intersection of cone projection circles on a plane.

3.3 Inscribed regular polygon intersection

Finding the common intersection of n circles on a plane has a time complexity of $O(n \log n)$ [34], which cannot be used for designing one-pass trajectory simplification algorithms using SED. Intuitively, we can approximate a circle with its m-edges inscribed polygon whose intersection can be computed much faster. However, the intersection of two general m-edges polygons may produce a polygon with more than m edges; thus, the time complexity of finding the common intersection of n polygons on a plane is not linear. To solve this problem, we approximate a circle with a fixed rotation and m-edges inscribed regular polygon.

Inscribed regular polygons ($\mathcal{R}$). Given a cone projection circle $\mathcal{O}^c(P, r)$, its inscribed m-edges regular polygon is denoted as $\mathcal{R}(V, E)$, where (1) $V = \{v_1, \ldots, v_m\}$ is the set of vertexes that are defined by a polar coordinate system, whose origin is the center P of $\mathcal{O}^c$, as follows:

$$v_j = (r, \frac{(j-1)}{m} 2\pi), \ \ j \in [1, m],$$

and (2) $E = \{\overrightarrow{v_m v_1}\} \bigcup \{\overrightarrow{v_j v_{j+1}} \mid j \in [1, m-1]\}$ is the set of edges that are labeled with the subscripts of their start points. Figure 8(1) illustrates the inscribed regular octagon ($m = 8$) of a cone projection circle $\mathcal{O}^c(P, r)$.

Let $\mathcal{R}_{s+i}$ ($1 \le i \le k$) be the inscribed regular polygon of the cone projection circle $\mathcal{O}^c(P^c_{s+i}, r^c_{s+i})$, $\mathcal{R}^*_l$ ($1 \le l \le k$) be the intersection $\bigcap_{i=1}^{l} \mathcal{R}_{s+i}$, and E^j ($1 \le j \le m$) be the group of k edges labeled with j in all $\mathcal{R}_{s+i}$ ($i \in [1, k]$). It is easy to verify that all edges in the same edge group E^j ($1 \le j \le m$) are in parallel (or overlapping) with each other by the above definition of inscribed regular polygons, as illustrated in Fig. 8(2).

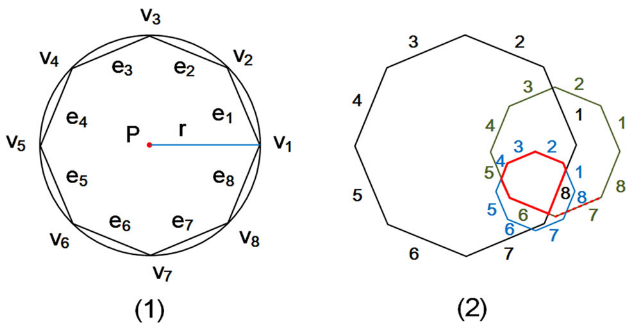

Fig. 8 Regular octagons and their intersections ($m = 8$)

The intersection of inscribed regular polygons holds a nice property, as shown below.

Proposition 3 *The intersection $\mathcal{R}_l^* \prod \mathcal{R}_{s+l+1}$ $(1 \le l < k)$ has at most m edges,* i.e., *at most one edge from each edge group.*

Figure 8(2) shows the intersection polygon (red lines) of $\mathcal{R}_1$, $\mathcal{R}_2$ and $\mathcal{R}_3$ with 7 edges, and here edges labeled with 7 have no contributions to the resulting intersection polygon.

Based on Proposition 3, we also have the following.

Proposition 4 *The intersection of $\mathcal{R}_l^*$ and $\mathcal{R}_{s+l+1}$ $(1 \le l < k)$ can be done in $O(1)$ time.*

By Proposition 4, we have a local synchronous distance checking method.

3.4 Speedup inscribed regular polygon intersection

Recall that the intersection of inscribed regular polygons can be computed by the convex polygon intersection algorithm CPolyInter [25] in Fig. 5. However, observe that algorithm CPolyInter is for general convex polygons, while the inscribed regular polygons $\mathcal{R}_{s+i}$ $(i \in [1, k])$ of the cone projection circles are constructed in a unified way, *i.e.,* fixed rotation and edge number, which allows us to develop a faster method to speed up the computation of their intersection.

We next explain the key idea for speeding up the computation. Observe that when the edges $\overrightarrow{A} = (P_{s_A}, P_{e_A})$ and $\overrightarrow{B} = (P_{s_B}, P_{e_B})$ on polygons $\mathcal{R}_2$ and $\mathcal{R}_1$ (both $\mathcal{R}_2$ and $\mathcal{R}_1$ can be either inscribed regular polygons or the common intersection of inscribed regular polygons) are "chasing" one another as the way the convex polygon intersection algorithm CPolyInter does, every segment in the two polygons being intersected has to originate from one of the m edges of the regular polygons, and consider the geometric similarity of the regular polygons, we may advance edge $\overrightarrow{A}$ or $\overrightarrow{B}$ multiple steps at a time, instead of a single step at a time as the convex polygon intersection algorithm CPolyInter does. For example, in Fig. 4(1)–(5), edge $\overrightarrow{A}$ successively moves four steps, each under the advance rule (1) "$(\overrightarrow{A} \times \overrightarrow{B} < 0$ and $P_{e_A} \notin \mathcal{H}(\overrightarrow{B}))$ or $(\overrightarrow{A} \times \overrightarrow{B} \ge 0$ and $P_{e_B} \in \mathcal{H}(\overrightarrow{A}))$" of algorithm CPolyInter. Alternatively, we can directly move A from Fig. 4(1) to (5), by reducing four steps to one step only. To achieve this, we first develop two extra *advance rules* (Propositions 5 and 6) for the intersection of inscribed regular polygons.

Proposition 5 *If either $(\overrightarrow{A} \prod \overrightarrow{B} \ne \emptyset$* **and** *$\overrightarrow{A} \times \overrightarrow{B} < 0$* **and** *$P_{e_A} \notin \mathcal{H}(\overrightarrow{B}))$ or $(\overrightarrow{A} \prod \overrightarrow{B} \ne \emptyset$* **and** *$\overrightarrow{A} \times \overrightarrow{B} \ge 0$* **and** *$P_{e_B} \in \mathcal{H}(\overrightarrow{A}))$ holds, then $\overrightarrow{A}$ advances s steps such that*

$$s = \begin{cases} 2 \times (g(\overrightarrow{B}) - g(\overrightarrow{A})) & if\ g(\overrightarrow{B}) > g(\overrightarrow{A}) \\ 1 & if\ g(\overrightarrow{A}) = g(\overrightarrow{B}) \\ 2 \times (m + g(\overrightarrow{B}) - g(\overrightarrow{A})) & if\ g(\overrightarrow{B}) < g(\overrightarrow{A}), \end{cases}$$

in which $g(e)$ denotes the label of edge e.

We next explain how the edge $\overrightarrow{A}$ advances based on Proposition 5. Indeed, $\overrightarrow{A}$ moves from its original position to its symmetric edge on $\mathcal{R}_{s+l+1}$ *w.r.t.* the symmetric line that is perpendicular to $\overrightarrow{B}$ on $\mathcal{R}_l^*$. For example, in Fig. 9(1), there is $\overrightarrow{A} \prod \overrightarrow{B} \ne \emptyset$ **and** $\overrightarrow{A} \times \overrightarrow{B} \ge 0$ **and** $P_{e_B} \in \mathcal{H}(\overrightarrow{A})$; hence, $\overrightarrow{A}$ moves on. As $g(\overrightarrow{B}) = 3 > 1 = g(\overrightarrow{A})$, $\overrightarrow{A}$ moves forward $2 \times (g(\overrightarrow{B}) - g(\overrightarrow{A})) = 2 \times (3 - 1) = 4$ steps. Here, the label of edge $\overrightarrow{A}$ is changed to 5, its symmetric edge 1 on $\mathcal{R}_{s+l+1}$ *w.r.t.* the symmetric line that is perpendicular to $\overrightarrow{B}$ labeled with 3 on $\mathcal{R}_l^*$.

Proposition 6 *If either $(\overrightarrow{A} \prod \overrightarrow{B} \ne \emptyset$* **and** *$\overrightarrow{A} \times \overrightarrow{B} \ge 0$* **and** *$P_{e_B} \notin \mathcal{H}(\overrightarrow{A}))$ or $(\overrightarrow{A} \prod \overrightarrow{B} \ne \emptyset$* **and** *$\overrightarrow{A} \times \overrightarrow{B} < 0$* **and** *$P_{e_A} \in \mathcal{H}(\overrightarrow{B}))$ holds, then edge $\overrightarrow{B}$ is directly moved to the edge after the one having the same edge group as edge $\overrightarrow{A}$.*

We next explain how the edge $\overrightarrow{B}$ advances based on Proposition 6. For example, in Fig. 9(2), $\overrightarrow{A} \prod \overrightarrow{B} \ne \emptyset$ **and** $\overrightarrow{A} \times \overrightarrow{B} < 0$ **and** $P_{e_A} \in \mathcal{H}(\overrightarrow{B})$, hence $\overrightarrow{B}$ moves forward. As the edge $\overrightarrow{A}$ is labeled with 7, $\overrightarrow{B}$ moves to the edge labeled with 8 on $\mathcal{R}_l^*$, which is the next of the edge labeled with 7 on $\mathcal{R}_l^*$. Note that if the edge labeled with 8 did not actually exist in the intersection polygon $\mathcal{R}_l^*$, then $\overrightarrow{B}$ should repeatedly move on until it reaches the first "real" edge on $\mathcal{R}_l^*$.

We then present our faster algorithm for computing the intersection of inscribed regular polygons that uses the advance rules in terms of Propositions 5 and 6 .

Algorithm FastRPolyInter. The regular polygon intersection algorithm, *i.e.,* FastRPolyInter, is the optimized version of the convex polygon intersection algorithm CPolyInter, based on Propositions 5 and 6 . We also save vertexes of a polygon in a fixed size array, which is different from CPolyInter that saves

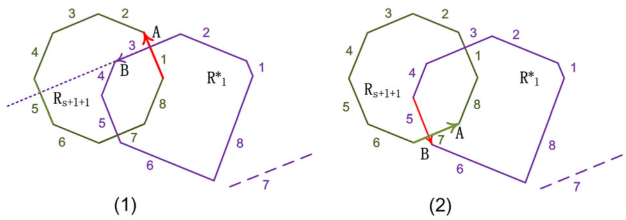

Fig. 9 Examples of fast advancing rules

polygons in linked lists. Considering the regular polygons each having a fixed number of vertexes/edges, marked from 1 to m, this policy allows us to quickly address an edge or a vertex with its label.

Given intersection polygon $\mathcal{R}^*_l$ of the preview l polygons and the next approximate polygon $\mathcal{R}_{s+l+1}$, algorithm FastRPolyInter returns $\mathcal{R}^*_{l+1} = \mathcal{R}^*_l \prod \mathcal{R}_{s+l+1}$. It runs the similar routine as the CPolyInter algorithm, except that (1) it saves polygons in arrays, and (2) the advance strategies are partitioned into two parts, *i.e.*, $\overrightarrow{A} \prod \overrightarrow{B} \neq \emptyset$ and $\overrightarrow{A} \prod \overrightarrow{B} = \emptyset$, where the former applies Propositions 5 and 6 , and the latter remains the same as algorithm CPolyInter.

Correctness and complexity analyses. Observe that algorithm FastRPolyInter basically has the same routine as algorithm CPolyInter, except that it speeds up the advance of directed edges $\overrightarrow{A}$ and $\overrightarrow{B}$ under certain circumstances as shown by Propositions 5 and 6 , which together ensure the correctness of FastRPolyInter. Moreover, algorithm FastRPolyInter runs in $O(1)$ time by Proposition 4.

4 One-pass trajectory simplification

A naive algorithm that directly outputs all the data points in the given trajectory is one-pass, however, it is hardly to be an effective trajectory simplification algorithm. Does there exist an effective, one-pass and error bounded line simplification algorithm using SED? This remains an open question in the community. We next present a positive answer to this question. The main result here is stated as follows.

Theorem 7 *There exist effective one-pass error bounded and strong and weak trajectory simplification algorithms using the synchronous Euclidean distance (*SED*).*

We shall prove this by providing such algorithms employing the synchronous distance checking technique developed in Sect. 3. More specifically, following [14,39], we consider two classes of trajectory simplification. The first one, referred to as *strong simplification*, that takes as input a trajectory $\dddot{\mathcal{T}}$, an error bound ϵ and the number m of edges for inscribed regular polygons, and produces a simplified trajectory $\dddot{\mathcal{T}}'$ such that all data points in $\dddot{\mathcal{T}}'$ belong to $\dddot{\mathcal{T}}$. The second one,

Algorithm CISED-S ($\dddot{\mathcal{T}}[P_0, \ldots, P_n]$, ϵ, m)
1. $P_s := P_0$; $i := 1$; $\mathcal{R}^* := \emptyset$; $\overline{\mathcal{T}} := \emptyset$; $t_c := P_1.t$;
2. **while** $i \le n$ **do**
3. $\quad\mathcal{R} :=$ getRegularPolygon $(P_s, P_i, \epsilon/2, m, t_c)$;
4. $\quad$**if** $\mathcal{R}^* = \emptyset$ **then** /* $\mathcal{R}^*$ needs to be initialized */
5. $\quad\quad\mathcal{R}^* := \mathcal{R}$;
6. $\quad$**else**
7. $\quad\quad\mathcal{R}^* :=$ FastRPolyInter($\mathcal{R}^*$, $\mathcal{R}$);
8. $\quad\quad$**if** $\mathcal{R}^* = \emptyset$ **then** /* generate a new line segment */
9. $\quad\quad\quad i := i - 1$; $\overline{\mathcal{T}} := \overline{\mathcal{T}} \cup \{\overrightarrow{P_s P_i}\}$;
10. $\quad\quad\quad P_s := P_i$; $t_c := P_{i+1}.t$;
11. $\quad i := i + 1$;
12. $\overline{\mathcal{T}} := \overline{\mathcal{T}} \cup \{\overrightarrow{P_s P_n}\}$;
13. **return** $\overline{\mathcal{T}}$.

Procedure getRegularPolygon (P_s, P_i, r, m, t_c)
1. $w := (t_c - t_s)/(P_i.t - P_s.t)$;
2. $x := P_s.x + w \cdot (P_i.x - P_s.x)$;
3. $y := P_s.y + w \cdot (P_i.y - P_s.y)$;
4. **for** $(j := 1; j \le m; j++)$ **do**
5. $\quad\theta := (2j + 1) * \pi/m$;
6. $\quad\mathcal{R}.v_j.x := x + w \cdot r \cdot \cos\theta$;
7. $\quad\mathcal{R}.v_j.y := y + w \cdot r \cdot \sin\theta$;
8. **return** $\mathcal{R}$.

Fig. 10 One-pass strong trajectory simplification algorithm

referred to as *weak simplification*, that takes as input a trajectory $\dddot{\mathcal{T}}$, an error bound ϵ and the number m of edges for inscribed regular polygons, and produces a simplified trajectory $\dddot{\mathcal{T}}'$ such that some data points in $\dddot{\mathcal{T}}'$ may not belong to $\dddot{\mathcal{T}}$. That is, weak simplification allows data interpolation.

4.1 Strong trajectory simplification

Recall that in Propositions 1 and 2 , the point Q may not be in the input sub-trajectory $[P_s, \ldots, P_{s+k}]$. If we restrict $Q = P_{s+k}$, the end point of the sub-trajectory, then the narrow cones whose base circles with a radius of $\epsilon/2$ suffice, as shown below.

Proposition 8 *Given a sub-trajectory* $[P_s, \ldots, P_{s+k}]$ *and an error bound* ϵ, $sed(P_{s+i}, \overrightarrow{P_s P_{s+k}}) \le \epsilon$ *for each* $i \in [1, k]$ *if* $\prod_{i=1}^{k} \mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon/2)) \neq \{P_s\}$.

We now present the one-pass error bounded *strong trajectory simplification* algorithm using SED based on Proposition 8, as shown in Fig. 10.

Procedure getRegularPolygon. We first present procedure getRegularPolygon that, given a cone projection circle, generates its inscribed m-edges regular polygon, following the definition in Sect. 3.3.

The parameters P_s, P_i, r and t_c together form the projection circle $\mathcal{O}^c(P^c_i, r^c_i)$ of the spatiotemporal cone $\mathcal{C}(P_s, \mathcal{O}(P_i, r))$ of point P_i *w.r.t.* point P_s on the plane $P.t - t_c = 0$. Firstly, $P^c_i.x$ and $P^c_i.y$ are computed (lines 1–3), and $r^c_i = w \cdot r$. Then, it builds and returns an m-edges inscribed

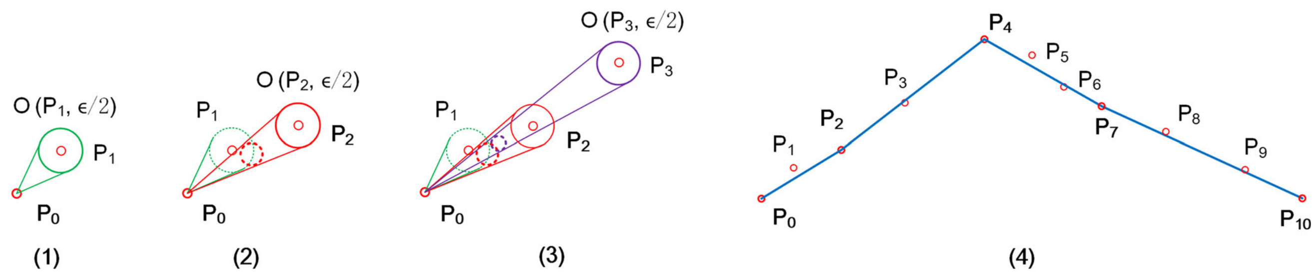

Fig. 11 A running example of the CISED-S algorithm. The points and the oblique circular cones are projected on an x-y space. The trajectory $\dddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ is compressed into four line segments. The solid line circles are *synchronous circles*, each has a radius of $\epsilon/2$, and the dash line circles are *cone projection circles* whose inscribe regular polygons are computed

regular polygon $\mathcal{R}$ of $\mathcal{O}^c(P_i^c, r_i^c)$ (lines 4–8), by transforming a polar coordinate system into a Cartesian one. Note here θ, $r \cdot \sin\theta$ and $r \cdot \cos\theta$ only need to be computed once during the entire processing of a trajectory.

Algorithm CISED-S. It takes as input a trajectory $\dddot{\mathcal{T}}[P_0, \ldots, P_n]$, an error bound ϵ and the number m of edges for inscribed regular polygons, and returns a simplified trajectory $\overline{\mathcal{T}}$ of $\dddot{\mathcal{T}}$.

The algorithm first initializes the start point P_s to P_0, the index i of the current data point to 1, the intersection polygon $\mathcal{R}^*$ to $\emptyset$, the output $\overline{\mathcal{T}}$ to $\emptyset$, and t_c to $P_1.t$, respectively (line 1). The algorithm sequentially processes the data points of the trajectory one by one (lines 2–10). It gets the m-edge inscribed regular polygon *w.r.t.* the current point P_i (line 3) by calling procedure getRegularPolygon. When $\mathcal{R}^* = \emptyset$, the intersection polygon $\mathcal{R}^*$ is simply initialized as $\mathcal{R}$ (lines 4, 5). Otherwise, $\mathcal{R}^*$ is the intersection of the current regular polygon $\mathcal{R}$ with $\mathcal{R}^*$ by calling procedure FastRPolyInter() introduced in Sect. 3.4 (line 7). If the resulting intersection $\mathcal{R}^*$ is empty, then a new line segment $\overrightarrow{P_s P_{i-1}}$ is generated (lines 8–10). The process repeats until all points have been processed (line 11). After the final line segment $\overrightarrow{P_s P_n}$ is generated (line 12), it returns the simplified piecewise line representation $\overline{\mathcal{T}}$ (line 13).

Example 6 Figure 11 shows a running example of CISED-S for compressing the trajectory $\dddot{\mathcal{T}}$ in Fig. 2.

(1) After initialization, the CISED-S algorithm reads point P_1 and builds a narrow *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). The *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). The *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 (red) is returned. As $\mathcal{R}^* = \mathcal{R}_1$ is not empty, $\mathcal{R}^*$ is set to the intersection of $\mathcal{R}_2$ and $\mathcal{R}^*$, which is $\mathcal{R}_1 \bigcap \mathcal{R}_2 \neq \emptyset$.

(3) For point P_3, the algorithm runs the same routine as P_2 until the intersection of $\mathcal{R}_3$ and $\mathcal{R}^*$ is $\emptyset$. Thus, a line segment $\overrightarrow{P_0 P_2}$ 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.*, $\{\overrightarrow{P_0 P_2}, \overrightarrow{P_2 P_4}, \overrightarrow{P_4 P_7}, \overrightarrow{P_7 P_{10}}\}$ (Fig. 10). □

4.2 Weak trajectory simplification

We then present the one-pass error bounded *weak simplification* algorithm using SED. By allowing data interpolations, it extends the radii of the base circles of spatiotemporal cones in CISED-S from $\epsilon/2$ to ϵ, which leads to better compression ratios than CISED-S. Recall that in Proposition 2, the point Q may not be in the input sub-trajectory $[P_s, \ldots, P_{s+k}]$, which can be treated as an interpolated data point.

Algorithm CISED-W. Given a trajectory $\dddot{\mathcal{T}}[P_0, \ldots, P_n]$, an error bound ϵ and the number m of edges for inscribed regular polygons, it returns a simplified trajectory, which may contain interpolated points. By Proposition 2, algorithm CISED-W generates spatiotemporal cones whose bases are circles with a radius of ϵ, and, hence, it replaces $\epsilon/2$ with ϵ (line 3 of CISED-S). It also generates new line segments with data points Q (may be interpolated points), and, hence, it replaces point P_i and line segment $\overrightarrow{P_s P_i}$ (lines 9 and 10 of algorithm CISED-S) with Q and $\overrightarrow{P_s Q}$, respectively, such that Q is generated as follows.

Proposition 9 *Given a sub-trajectory $\dddot{\mathcal{T}}[P_s, \ldots, P_{s+k}]$ and an error bound ϵ, $t_c = P_{s+k}.t$ and $\mathcal{R}_k^*$ be the intersection of all polygons $\mathcal{R}_{s+i}$ ($i \in [1, k]$) on the plane $P.t - t_c = 0$. If $\mathcal{R}_k^*$ is not empty, then any point in the area of $\mathcal{R}_k^*$ is feasible for Q.*

The choice of a point Q from $\mathcal{R}_k^*$ may slightly affect the compression ratios and average errors. However, the choice of an optimal Q is non-trivial. For the benefit of running time, we apply the following strategies.

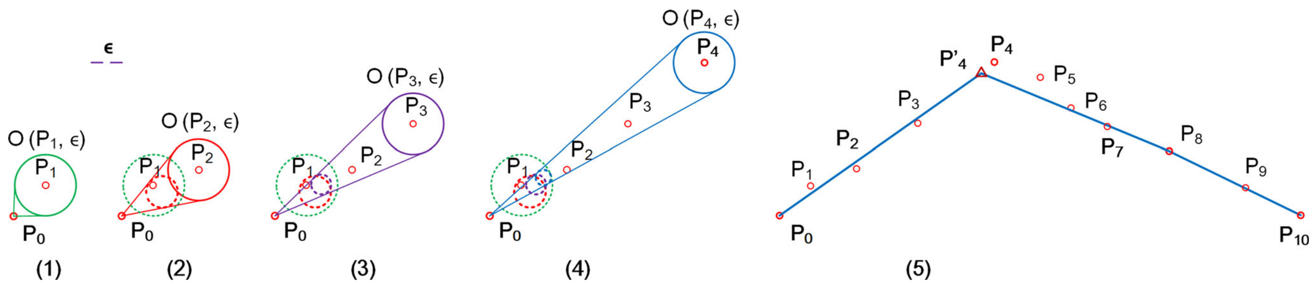

Fig. 12 A running example of the CISED-W algorithm. The points and the oblique circular cones are projected on an x-y space. The trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ is compressed into three line segments. The solid line circles are *synchronous circles*, each has a radius of ϵ, and the dash line circles are *cone projection circles*. Note in (5) a point, P'_4, is Interpolated

(1) If P_{s+k} is in the area of $\mathcal{R}^*_k$ *w.r.t.* $t_c = P_{s+k}.t$, then Q is simply set to P_{s+k}.

(2) If $\mathcal{R}^*_k \neq \emptyset$ and P_{s+k} is not in the area of $\mathcal{R}^*_k$ *w.r.t.* $t_c{=}P_{s+k}.t$, then the central point of $\mathcal{R}^*_k$ is chosen as Q.

(3) If $t_c \neq P_{s+k}.t$, which is the general case, then we project the intersection polygon $\mathcal{R}^*_k$ *w.r.t.* $t_c \neq P_{s+k}.t$ on the plane $P.t - P_{s+k}.t = 0$, and apply strategies (1) and (2) above. That is, the projection has no affects on the choice of Q.

Example 7 Figure 12 shows a running example of algorithm CISED-W for compressing the trajectory $\ddot{\mathcal{T}}$ in Fig. 2 again.

(1) After initialization, algorithm CISED-W reads point P_1 and builds an *oblique circular cone* $\mathcal{C}(P_0, \mathcal{O}(P_1, \epsilon))$, and projects it on the plane $P.t - P_1.t = 0$. The inscribed regular polygon $\mathcal{R}_1$ of the projection circle is returned, and the intersection $\mathcal{R}^*$ is set to $\mathcal{R}_1$.

(2) P_2, P_3 and P_4 are processed in turn. The intersection polygons $\mathcal{R}^*$ are not empty.

(3) For point P_5, the intersection of polygons $\mathcal{R}_5$ and $\mathcal{R}^*$ is $\emptyset$. Thus, line segment $\overrightarrow{P_0 Q} = \overrightarrow{P_0 P'_4}$ is output, and a new line segment is started such that point $Q = P'_4$ is the new start point and plane $P.t - P_5.t = 0$ is the new projection plane.

(4) At last, the algorithm outputs 3 continuous line segments, *i.e.*, $\overrightarrow{P_0 P'_4}$, $\overrightarrow{P'_4 P_8}$ and $\overrightarrow{P_8 P_{10}}$, in which P'_4 is an interpolated data point not in $\ddot{\mathcal{T}}$. □

Correctness and complexity analyses. The correctness of algorithms CISED-S and CISED-W follows from Propositions 2 and 8 , and Propositions 2 and 9 , respectively. It is easy to verify that each data point in a trajectory is only processed once, and each can be done in $O(1)$ time, as both procedures getRegularPolygon and FastRPolyInter can be done in $O(1)$ time. Hence, these algorithms are both one-pass error bounded trajectory simplification algorithms. It is also easy to see that these algorithms take $O(1)$ space.

5 Experimental study

In this section, we present an extensive experimental study of our one-pass trajectory simplification algorithms (CISED-S and CISED-W) compared with the compression optimal algorithm using SED (C-Optimal) and existing algorithms of DPSED and SQUISH-E on trajectory datasets. Using four real-life trajectory datasets, we conducted sets of experiments to evaluate: (1) the compression ratios of algorithms CISED-S and CISED-W vs. DPSED, SQUISH-E and C-Optimal, (2) the average errors of algorithms CISED-S and CISED-W vs. DPSED, SQUISH-E and C-Optimal, (3) the running time of algorithms CISED-S and CISED-W vs. DPSED, SQUISH-E and C-Optimal, (4) the impacts of polygon intersection algorithms FastRPolyInter and CPolyInter and the edge number m of inscribed regular polygons to the compression ratios, errors and running time of algorithms CISED-S and CISED-W, (5) the impacts of the distance metrics PED and SED on the compression ratios, errors and running time of trajectory simplification algorithms and (6) the impacts of the distance metrics PED and SED on spatiotemporal queries.

5.1 Experimental setting

Real-life Trajectory Datasets. We use four real-life datasets ServiceCar, GeoLife, Mopsi and PrivateCar shown in Table 2 to test our solutions.

(1) *Service car trajectory data* (ServiceCar) is the GPS trajectories collected by a Chinese car rental company during April 2015 to November 2015. The sampling rate was one point per 3–5 s, and each trajectory has around $114.1K$ points.

(2) *GeoLife trajectory data* (GeoLife) is the GPS trajectories collected in GeoLife project [43] by 182 users in a period from Apr. 2007 to Oct. 2011. These trajectories have a variety of sampling rates, among which 91% are logged with one point per 1–5 s.

(3) *Mopsi trajectory data* (Mopsi) is the GPS trajectories collected in Mopsi project [21] by 51 users in a period from 2008 to 2014. Most routes are in Joensuu region, Finland.

Table 2 Real-life trajectory datasets

Data sets	Number of trajectories	Sampling rates (s)	Points per trajectory (K)	**Total points**
ServiceCar	1,000	3–5	$\sim$ 114.0	114M
GeoLife	182	1–5	$\sim$ 131.4	24.2M
Mopsi	51	2	$\sim$ 153.9	7.9M
PrivateCar	10	1	$\sim$ 11.8	112.8K

The sampling rate was one point per 2 s, and each trajectory has around 153.9K points.
(4) *Private car trajectory data* (PrivateCar) is a small set GPS trajectories collected with a high sampling rate of one point per second by our team members in 2017. There are 10 trajectories and each trajectory has around 11.8K points.

As the compression optimal algorithm C-Optimal[13] has both high time and space complexities, *i.e.,* $O(n^3)$ time and $O(n^2)$ space, it is impossible to compress the entire datasets (slow and out of memory). Hence, we further build four *small datasets* such that each includes 10 middle-size (10K points per trajectory) trajectories selected from ServiceCar, GeoLife, Mopsi and PrivateCar, respectively.

Algorithms and implementation. We implement seven LS algorithms, *i.e.,* our CISED-S and CISED-W, sector intersection algorithm using PED (SIPED) [7,42], DPPED and DPSED (DP using PED [6] and DP using SED [19], the existing sub-optimal LS algorithms having the best compression ratios), SQUISH-E [23] (the fastest existing LS algorithm using SED) and C-Optimal (the compression optimal LS algorithm using SED, see Sect. 2.2). We also implement the polygon intersection algorithms, CPolyInter and our FastRPolyInter.

All algorithms were implemented with Java. All tests were run on an x64-based PC with 8 Intel(R) Core(TM) i7-6700 CPU @ 3.40GHz and 8GB of memory, and each test was repeated over 3 times and the average is reported here.

5.2 Experimental results

We next present our findings.

5.2.1 Evaluation of compression ratios

In the first set of tests, we evaluate the impacts of parameter m on the compression ratios of our algorithms CISED-S and CISED-W, and compare the compression ratios of CISED-S and CISED-W with DPSED, SQUISH-E and C-Optimal. The compression ratio is defined as follows: Given a set of trajectories $\{\ddot{\mathcal{T}}_1, \ldots, \ddot{\mathcal{T}}_M\}$ and their piecewise line representations $\{\overline{\mathcal{T}}_1, \ldots, \overline{\mathcal{T}}_M\}$, the compression ratio of an algorithm is $(\sum_{j=1}^{M} |\overline{\mathcal{T}}_j|)/(\sum_{j=1}^{M} |\ddot{\mathcal{T}}_j|)$. By the definition, *algorithms with lower compression ratios are better.*

Exp-1.1: Impacts of parameter m on compression ratios. To evaluate the impacts of the number m of edges of polygons on the compression ratios of algorithms CISED-S and CISED-W, and also to confirm that our fast regular polygon intersection algorithm FastRPolyInter has the same compression ratios as the convex polygon intersection algorithm CPolyInter, we fixed the error bound $\epsilon = 60$ m, and varied m from 4 to 40. The results are reported in Fig. 13.
(1) Algorithms CISED-S and CISED-W using FastRPolyInter have the same compression ratios as their counterparts using CPolyInter for all cases.
(2) When varying m, the compression ratios of algorithms CISED-S and CISED-W decrease with the increase of m on all datasets. The increase of edge number m of a regular polygon makes the polygon better approximate its corresponding circle, which leads to a higher potential to have common intersections, and has a better compression ratio.
(3) When varying m, the compression ratios of algorithms CISED-S and CISED-W decrease (a) fast when $m < 12$, (b) slowly when $m \in [12, 20]$, and (c) very slowly when $m > 20$. Hence, *the region of* [12, 20] *is a good candidate region for m in terms of compression ratios.* Here, the compression ratio of m=12 is only on average 100.95% of m=20.

Exp-1.2: Impacts of the error bound ϵ on compression ratios (vs. algorithms DPSED and SQUISH-E). To evaluate the impacts of error bound ϵ on compression ratios, we fixed m=16, the middle of [12, 20], and varied ϵ from 10 to 200 m on the entire four datasets, respectively. The results are reported in Fig. 14 .
(1) When increasing ϵ, the compression ratios of all these algorithms decrease on all datasets, as it is clear that a larger ϵ makes more points represented by a line segment, and brings a better compression ratio.
(2) Dataset PrivateCar has the lowest compression ratios, compared with datasets Mopsi, ServiceCar and GeoLife, due to its highest sampling rate, ServiceCar has the highest compression ratios due to its lowest sampling rate, and GeoLife and Mopsi have the compression ratios in the middle accordingly.
(3) Algorithm CISED-S is better than SQUISH-E and close to DPSED on all datasets and for all ϵ. The compression ratios of CISED-S are on average (79.3%, 71.9%,

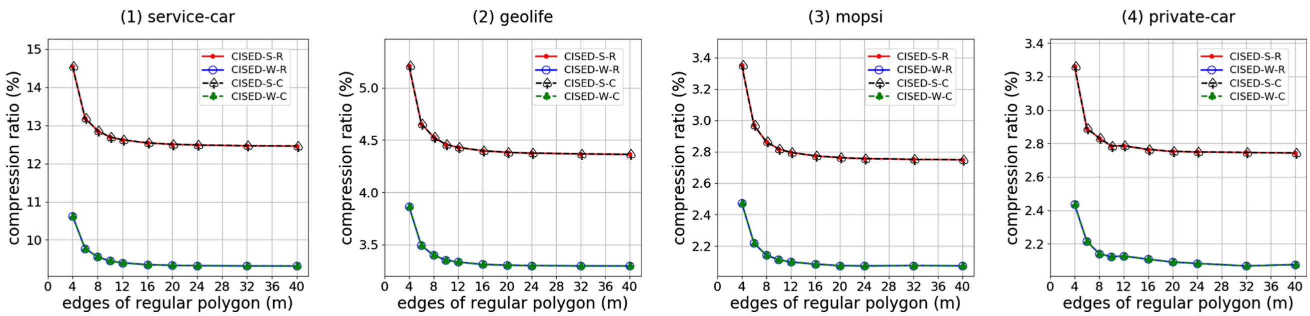

Fig. 13 Evaluation of compression ratios: fixed error bound with $\epsilon = 60$ m and varying m. Here, "R" denotes our fast regular polygon intersection algorithm FastRPolyInter, and "C" denotes the convex polygon intersection algorithm CPolyInter, respectively

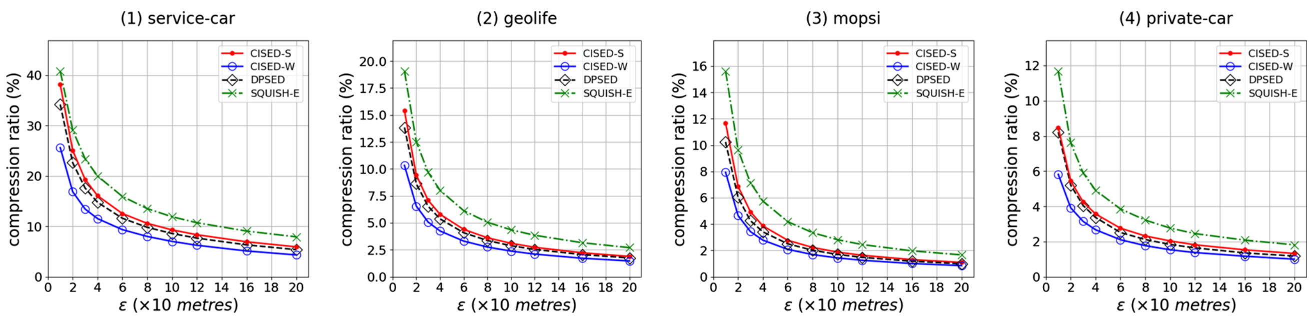

Fig. 14 Evaluation of compression ratios: fixed with $m = 16$ and varying error bound ϵ

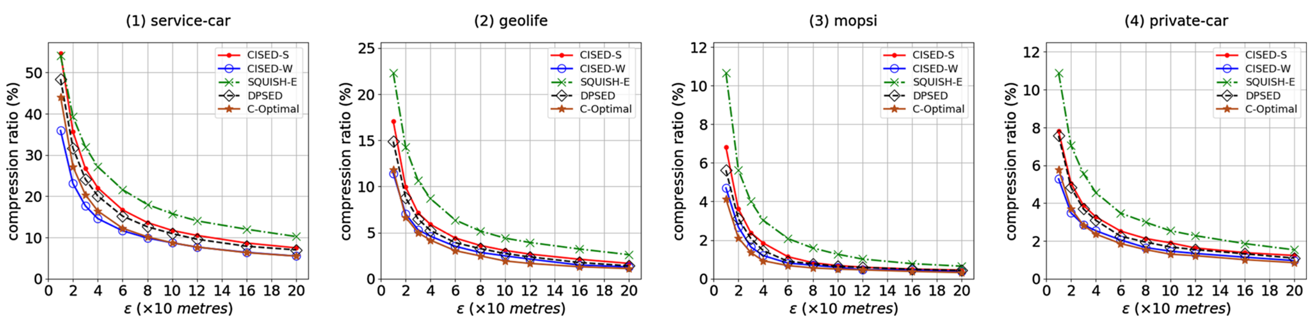

Fig. 15 Evaluation of compression ratios: fixed with $m = 16$ and varying error bound ϵ (on small datasets)

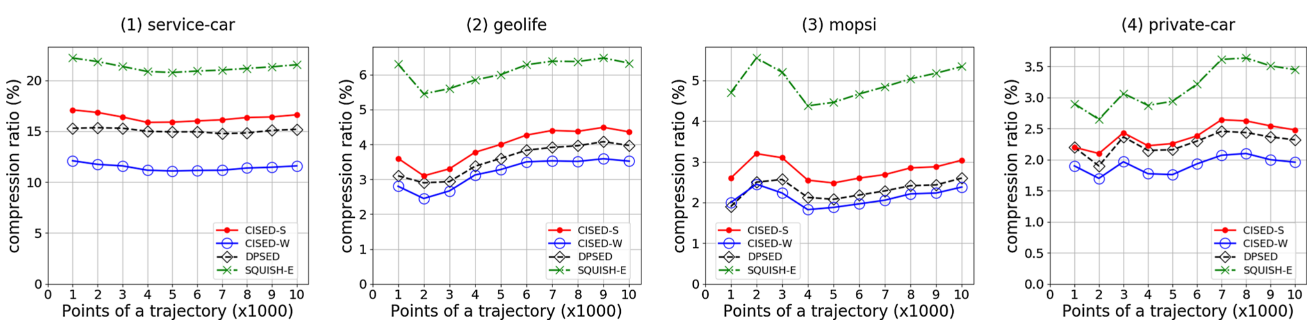

Fig. 16 Evaluation of compression ratios: fixed with $m = 16$ and $\epsilon = 60$ m, and varying the size of trajectories

67.3%, 72.7%) and (109.2%, 108.0%, 111.7%, 109.1%) of SQUISH-E and DPSED on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. For example, when ϵ = 40 m, the compression ratios of algorithms SQUISH-E, CISED-S and DPSED are (20.0%, 8.0%, 5.7%, 4.9%), (16.1%, 5.8%, 3.9%, 3.6%) and (14.8%, 5.4%, 3.4%, 3.4%) on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. CISED-S shows better compression ratios than

SQUISH-E because CISED-S applies an approximate policy that includes as many points as possible into a line segment during the process, while SQUISH-E applies a loose error prediction policy, which may ignore too many potential points that could be represented by a line segment in order to assure the error bound.

(4) Algorithm CISED-W has better compression ratios than DPSED, SQUISH-E and CISED-S on all datasets and for all ϵ. The compression ratios of CISED-W are on average (57.7%, 53.8%, 50.0%, 54.6%), (79.5%, 81.0%, 83.0%, 82.0%) and (72.9%, 75.0%, 74.3%, 75.1%) of algorithms SQUISH-E, DPSED and CISED-S on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. For example, when ϵ = 40 m, the compression ratios of algorithm CISED-W are (11.5%, 4.3%, 2.8%, 2.7%) on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. Algorithm CISED-W extends the radii of base circles of spatiotemporal cones from $\epsilon/2$ in CISED-S to ϵ, thus, it has better compression ratios than CISED-S.

Exp-1.3: Impacts of the error bound ϵ on compression ratios (vs. the compression optimal algorithm). To evaluate the impacts of error bound ϵ on compression ratios, we again fixed m=16, the middle of [12, 20], and varied ϵ from 10 to 200 m on the first $1K$ points of each trajectory of the selected *small datasets*, respectively. The results are reported in Fig. 15 .

(1) Algorithm CISED-S is worse than the compression optimal algorithm C-Optimal on all datasets and for all ϵ. More specifically, the compression ratios of CISED-S are on average (134.6%, 150.7%, 155.5%, 138.5%) of C-Optimal on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. For example, when ϵ = 40 m, the compression ratios of CISED-S and C-Optimal are (22.0%, 5.9%, 1.9%, 3.3%) and (16.4%, 4.2%, 0.9%, 2.4%) on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively.

(2) Algorithm CISED-W has the closest compression ratios to the compression optimal algorithm C-Optimal on all datasets and for all ϵ. The compression ratios of CISED-W are on average (94.8%, 115.5%, 119.7%, 107.5%) of C-Optimal on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. For example, when ϵ = 40 m, the compression ratios of algorithm CISED-W are (14.6%, 4.6%, 1.2%, 2.5%) on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. This is because algorithm CISED-W allows data interpolations, by Proposition 2, it extends the radii of the base circles of the spatiotemporal cones from $\epsilon/2$ in CISED-S to ϵ in CISED-W to contain more points.

Exp-1.4: Impacts of trajectory sizes on compression ratios. To evaluate the impacts of trajectory size, *i.e.,* the number of data points in a trajectory, on compression ratios, we chose the same 10 trajectories from datasets ServiceCar, GeoLife, Mopsi and PrivateCar, respectively, fixed m=16 and ϵ=60 m, and varied the size $|\ddot{\mathcal{T}}|$ of trajectories from $1K$ points to $10K$ points. The results are reported in Fig. 16.

(1) The compression ratios of these algorithms ordered from the best to the worst are CISED-W, DPSED, CISED-S and SQUISH-E, on all datasets with all sizes of trajectories, which is consistent with the previous tests.

(2) The sizes of input trajectories have few impacts on the compression ratios of LS algorithms on all datasets.

5.2.2 Evaluation of average errors

In the second set of tests, we first evaluate the impacts of parameter m on the average errors of algorithms CISED-S and CISED-W, then compare the average errors of our algorithms CISED-S and CISED-W with DPSED, SQUISH-E and the compression optimal algorithm C-Optimal.

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 point $P_{j,i}$ 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 $\sum_{j=1}^{M}\sum_{i=0}^{M} d(P_{j,i}, \mathcal{L}_{l,i})/\sum_{j=1}^{M}|\ddot{\mathcal{T}}_j|$.

Exp-2.1: Impacts of parameter m on average errors. To evaluate the impacts of parameter m on average errors of algorithms CISED-S and CISED-W, and to confirm that our fast regular polygon intersection algorithm FastRPolyInter has the same average errors as the convex polygon intersection algorithm CPolyInter, we fixed the error bound ϵ = 60 m, and varied m from 4 to 40. The results are reported in Fig. 17.

(1) Algorithms CISED-S and CISED-W using FastRPolyInter have the same average errors as their counterparts using CPolyInter, respectively, on all datasets and for all m.

(2) When varying m, the average errors of algorithms CISED-S and CISED-W increase with the increase of m on all datasets. The increase of edge number m of a regular polygon makes the polygon more closely approximate its corresponding circle, which means that some points having larger SED, *i.e.,* closer to half-ϵ in CISED-S or ϵ in CISED-W, are also included to a line segment, which further leads to a larger average error.

(3) When varying m, similar to compression ratios, the average errors of algorithms CISED-S and CISED-W increase (a) fast when m < 12, (b) slowly when $m \in [12, 20]$, and (c) very slowly when m > 20. *The range of* [12, 20] *is also the good candidate region for m in terms of errors.* Here, the average error of m = 12 is only on average 98.49% of m = 20.

Exp-2.2: Impacts of the error bound ϵ on average errors (vs. algorithms DPSED and SQUISH-E). To evaluate the average errors of these algorithms, we fixed m=16, and varied ϵ from 10 to 200 m on the entire datasets ServiceCar, GeoLife, Mopsi and PrivateCar, respectively. The results are reported in Fig. 18.

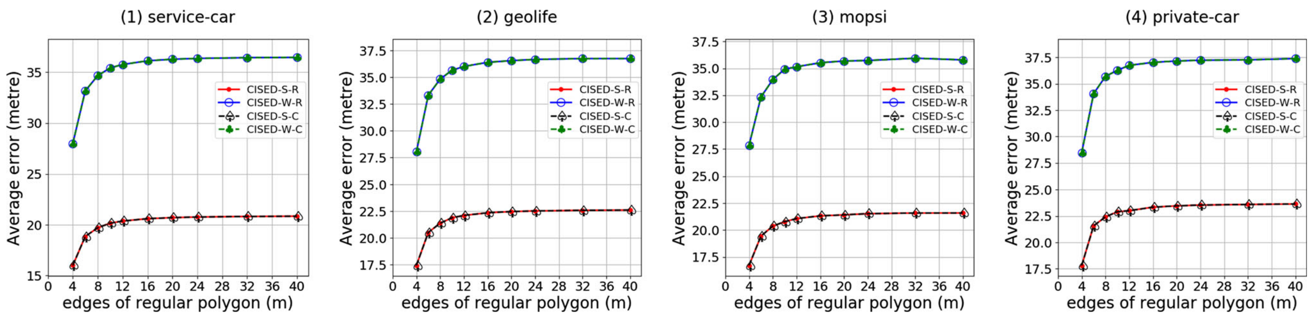

Fig. 17 Evaluation of average errors: fixed error bound with $\epsilon = 60$ m and varying m. Here, "R" denotes our fast regular polygon intersection algorithm FastRPolyInter, and "C" denotes CPolyInter, respectively

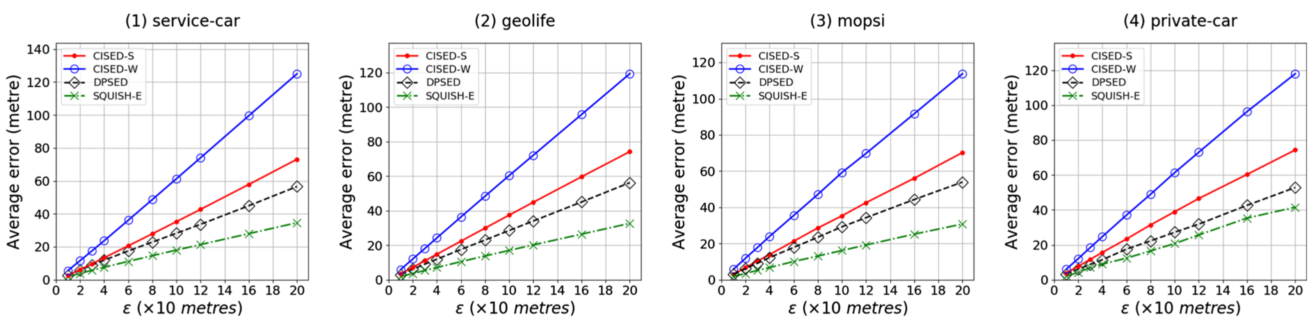

Fig. 18 Evaluation of average errors: fixed with $m = 16$ and varying error bound ϵ

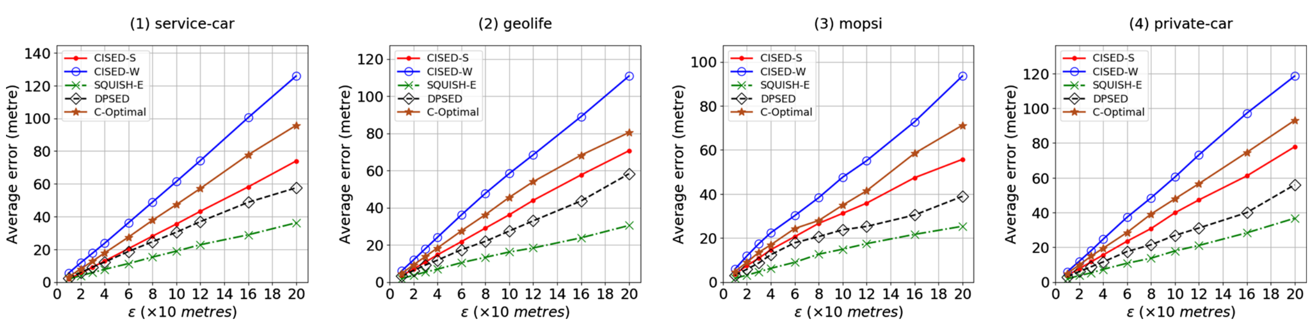

Fig. 19 Evaluation of average errors: fixed with $m = 16$ and varying error bound ϵ (on small datasets)

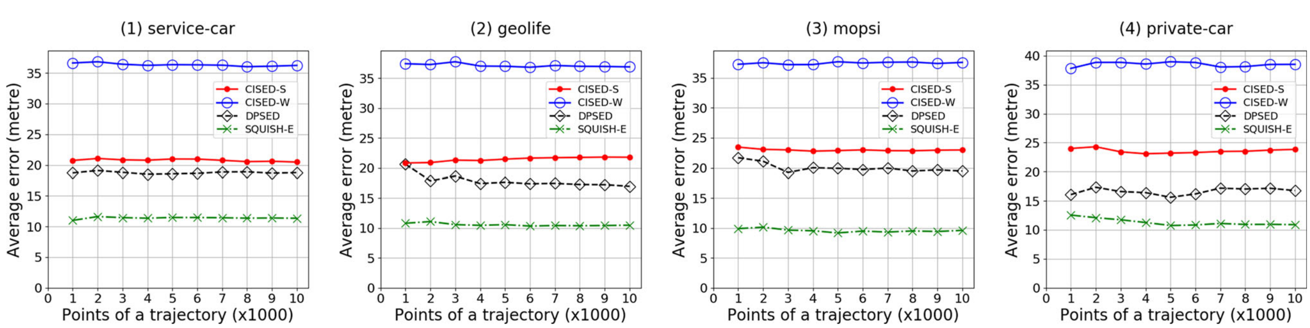

Fig. 20 Evaluation of average errors: fixed with $m = 16$ and $\epsilon = 60$ m, and varying the size of trajectories

(1) Average errors increase with the increase of ϵ. More specifically, the average error of each algorithm has approximately a linear relation to ϵ. It is clear that a larger ϵ includes more points into a line segment, including points with larger SED, which brings a better compression ratio as well as a larger average error.

(2) The average errors of these algorithms from the largest to the smallest are CISED-W, CISED-S, DPSED and SQUISH-

E, on all datasets and for all ϵ. The average errors of algorithms CISED-S and CISED-W are on average (119.3%, 127.7%, 119.9%, 138.0%) and (210.1%, 207.5%, 200.9%, 217.5%) of DPSED and (188.2%, 215.2%, 212.8%, 180.3%) and (331.1%, 349.7%, 356.7%, 284.2%) of SQUISH-E on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. Algorithms CISED-W and CISED-S apply an approximate policy, *i.e.,* they include as many points as possible into a line segment, and this policy usually leads to a larger average error.
(3) When the error bound of algorithm CISED-W is set as the half of CISED-S, the average errors of CISED-W are on average (93.8%, 86.0%, 81.4%, 79.4%) of CISED-S on datasets (ServiceCar, GeoLife,Mopsi, PrivateCar), respectively, meaning that the large average errors of algorithm CISED-W are caused by its cone *w.r.t.* ϵ compared with the narrow cone *w.r.t.* $\epsilon/2$ of CISED-S.

Exp-2.3: Impacts of the error bound ϵ on average errors (vs. the compression optimal algorithm). To evaluate the average errors of these algorithms, we fixed m=16, and varied ϵ from 10 to 200 m on the first $1K$ points of each trajectory of the *small datasets*, respectively. The results are reported in Fig. 19.

The average errors of these algorithms from the largest to the smallest are CISED-W, the compression optimal algorithm C-Optimal and CISED-S, on all datasets and for all ϵ. The average errors of CISED-S and CISED-W are on average (73.6%, 80.7%, 85.1%, 81.0%) and (133.3%, 130.7%, 131.0%, 126.3%) of C-Optimal on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. Note that here algorithm C-Optimal has the worst average error among all strong line simplification algorithms, as algorithm C-Optimal is optimal in terms of compression ratio, and algorithms with better compression ratios generate less line segments, which usually leads to larger errors.

Exp-2.4: Impacts of trajectory sizes on average errors. To evaluate the impacts of trajectory sizes on average errors, we chose the same 10 trajectories from datasets ServiceCar, GeoLife, Mopsi and PrivateCar, respectively. We fixed m=16 and $\epsilon = 60$ m, and varied the size $|\dddot{\mathcal{T}}|$ of trajectories from $1K$ points to $10K$ points. The results are reported in Fig. 20.
(1) The average errors of these algorithms ordered from the smallest to the largest are SQUISH-E, DPSED, CISED-S and CISED-W, on all datasets and for all trajectory sizes, which is consistent with the above tests.
(2) The sizes of input trajectories have few impacts on the average errors of LS algorithms on all datasets.

5.2.3 Evaluation of running time

In the third set of tests, we evaluate the impacts of parameter m on the running time of algorithms CISED-S and CISED-W, and compare the running time of our approaches CISED-S and CISED-W with algorithms C-Optimal, DPSED and SQUISH-E.

Exp-3.1: Impacts of algorithm FastRPolyInter and parameter m on running time. To evaluate the impacts of parameter m on the running time of algorithm CISED-S and CISED-W, and also to confirm that our fast regular polygon intersection algorithm FastRPolyInter runs faster than the convex polygon intersection algorithm CPolyInter, we equipped CISED-S and CISED-W with FastRPolyInter and CPolyInter, respectively, fixed $\epsilon = 60$ m, and varied m from 4 to 40. The results are reported in Figs. 21 and 22 .
(1) Algorithms CISED-S and CISED-W spend most their time on executing the polygon intersections. For all m, the execution time of algorithms CPolyInter and FastRPolyInter is on average (93.5%, 96.0%, 94.5%, 92.0%) and (90.5%, 92.5%, 91.0%, 90.5%) of the entire compression time on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively.
(2) FastRPolyInter runs faster than CPolyInter on all datasets and for all m due to the techniques that it applies to speed up the computation of polygon intersection. The execution time of algorithms CISED-S-FastRPolyInter and CISED-W-FastRPolyInter is one average 83.74% their counterparts with CPolyInter.
(3) When varying m, the execution time of algorithms CISED-S-FastRPolyInter, CISED-S-CPolyInter, CISED-W-FastRPolyInter and CISED-W-CPolyInter increases approximately linearly with the increase of m on all the datasets, *e.g.,* the running time of $m = 12$ is on average 69.92% of $m = 20$ for CISED-S and CISED-W on all datasets. This is because the time complexities of FastRPolyInter and CPolyInter are both $O(m)$, and a larger m leads to more comparisons of edges during the computation of polygon intersection.

Exp-3.2: **Impacts of the error bound ϵ on running time (VS. algorithms DPSED and SQUISH-E**). To evaluate the impacts of ϵ on running time, we fixed m = 16, and varied ϵ from 10 to 200 m on the entire datasets, respectively. The results are reported in Fig. 23.
(1) All algorithms are not very sensitive to ϵ on any datasets, and algorithm DPSED is more sensitive to ϵ than the other three algorithms. The running time of DPSED decreases a little bit with the increase of ϵ, as the increment of ϵ decreases the number of partitions of the input trajectory.
(2) Algorithms CISED-S and CISED-W are obviously faster than DPSED and SQUISH-E for all cases. They are on average (14.21, 18.19, 17.06, 9.98) times faster than DPSED, and (2.84, 3.45, 3.69, 2.86) times faster than SQUISH-E on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. This is consistent with their time complexity analyses.

Exp-3.3: Impacts of the error bound ϵ on running time (VS. the compression optimal algorithm). To evaluate the impacts of ϵ on running time, we fixed m = 16, and varied

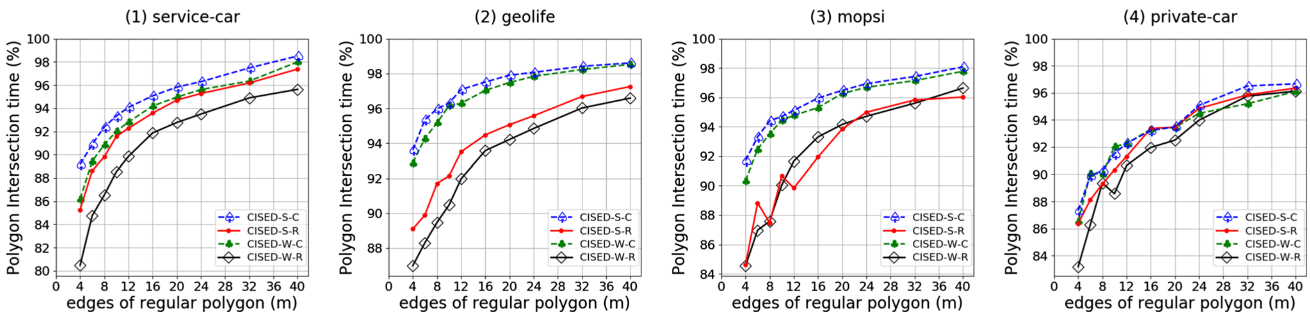

Fig. 21 Evaluation of running time of polygon intersection algorithms: fixed error bound with $\epsilon = 60$ m, and varying m. Here, "R" denotes our fast regular polygon intersection algorithm FastRPolyInter, and "C" denotes CPolyInter, respectively

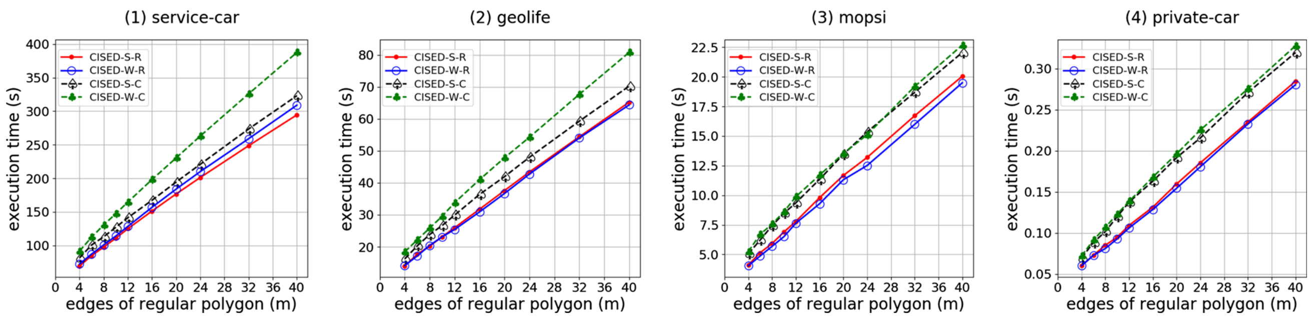

Fig. 22 Evaluation of running time: fixed error bound with $\epsilon = 60$ m, and varying m

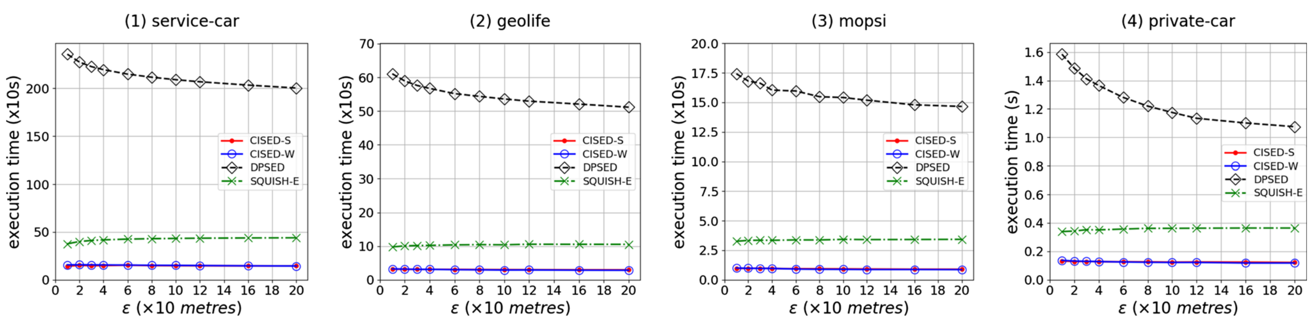

Fig. 23 Evaluation of running time: fixed with $m = 16$ and varying error bounds ϵ

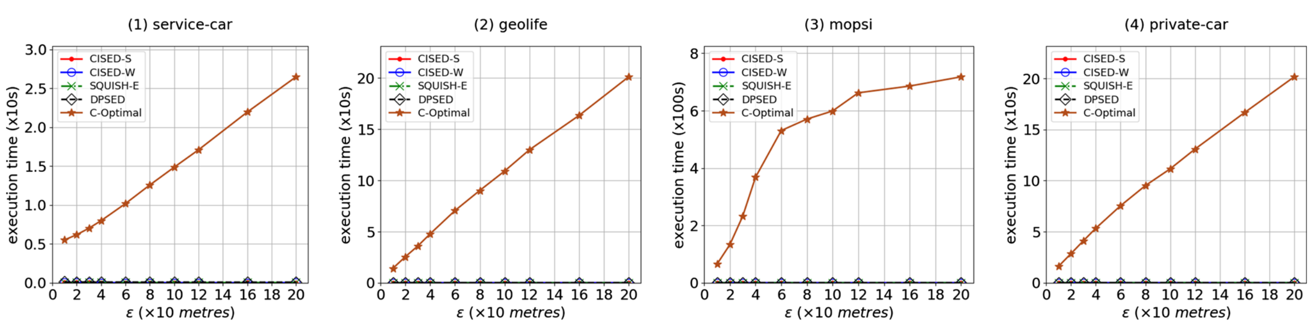

Fig. 24 Evaluation of running time: fixed with $m = 16$ and varying error bounds ϵ (on small datasets)

ϵ from 10 to 200 m on the first $1K$ points of each trajectory of the selected *small datasets*, respectively. The results are reported in Fig. 24.

(1) Algorithms CISED-S and CISED-W are obviously faster than C-Optimal for all cases. They are on average (925.25, 7888.26, 40041.59, 8528.76) times faster than C-Optimal on

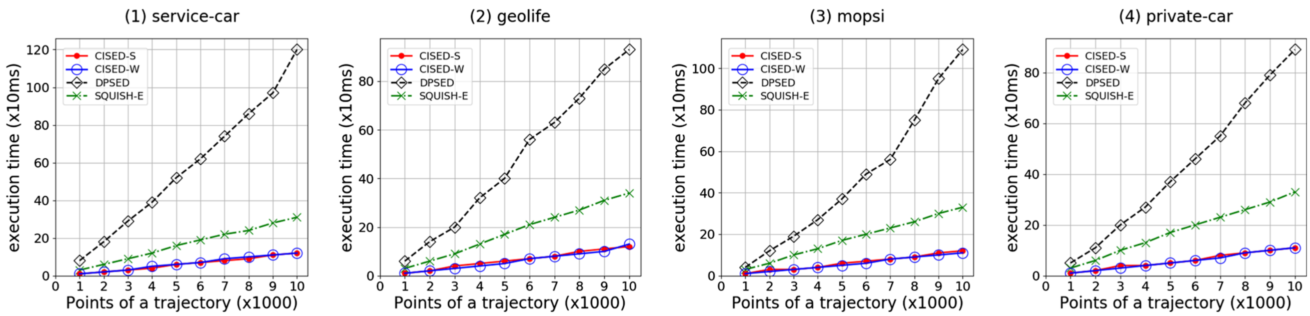

Fig. 25 Evaluation of running time: fixed with $m = 16$ and $\epsilon = 60$ m, and varying the size of trajectories

datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively.

Exp-3.4: Impacts of trajectory sizes on running time. To evaluate the impacts of trajectory sizes on running time, we chose the same 10 trajectories, from datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively, fixed $m = 16$ and $\epsilon = 60$ m, and varied the size $|\ddot{\mathcal{T}}|$ of trajectories from $1K$ points to $10K$ points. The results are reported in Fig. 25.

(1) Algorithms CISED-S and CISED-W are both the fastest LS algorithms using SED, and are (8.00–10.00, 5.83–8.11, 4.00–9.50, 5.00–8.09) times faster than DPSED, and (2.53–3.00, 2.62–3.12, 2.50–3.33, 2.89–3.40) times faster than SQUISH-E on the selected $1K$ to $10K$ points datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively.

(2) Algorithms CISED-S and CISED-W scale well with the increase of the size of trajectories on all datasets, and both have a linear running time, while algorithm DPSED does not. This is consistent with their time complexity analyses.

(3) The advantage of running time of algorithms CISED-S and CISED-W increases with the increase of trajectory sizes compared with DPSED and SQUISH-E.

5.2.4 Evaluation of distance metrics PED versus SED

In this set of tests, we compare the performance of algorithms using PED vs. SED. Two pairs of algorithms are tested, namely, (1) the algorithm DP using PED and SED, respectively, and (2) the sector intersection algorithm [36,40] using PED and our spatiotemporal cone intersection algorithm using SED.

Exp-4.1: Impacts of distance metrics on compression ratios. To evaluate the impacts of distance metrics, *i.e.,*PED and SED, on compression ratios, we fixed m=16 and varied ϵ from 10 to 200 m on the entire four datasets, respectively. The results are reported in Fig. 26.

Given the same error bound ϵ, the compression ratios using PED are obviously better than using SED. More specifically, the compression ratios of algorithm DP using PED are on average (47.1%, 55.5%, 60.7%, 44.7%) of algorithm DP using SED and the compression ratios of algorithm CISED-S are on average (45.4%, 54.5%, 60.1%, 43.0%) of algorithm SIPED on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. The reason behind this is that a PED of a point is the shortest distance from the point to a line segment, while a SED of a point is the distance between the point and its synchronized data point *w.r.t.* the line segment. Thus, the SED of a point to a line segment is always not less than the PED of the point to the line segment. Hence, given the same error bound ϵ, LS algorithms using SED usually include less points into a line segment. In other words, they generate more line segments.

Exp-4.2: Impacts of distance metrics on average errors. To evaluate the impacts of distance metrics on average errors, we fixed m=16 and varied ϵ from 10 to 200 m on the entire four datasets, respectively. The results are reported in Fig. 27.

Given the same error bound ϵ, the average errors of algorithms using SED are a bit larger than using PED. The average errors of algorithm DP using PED are on average (76.7%, 77.6%, 79.7%, 63.0%) of algorithm DP using SED, and the average errors of algorithm SIPED are on average (97.5%, 78.1%, 92.4%, 74.2%) of algorithm CISED-S on (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. As we know, 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. It seems that the speed factor introduces an extra error, and leads to a larger average error.

Exp-4.3: Impacts of distance metrics on running time. To evaluate the impacts of distance metrics on running time, we also fixed m=16 and varied ϵ from 10 to 200 m on the entire four datasets, respectively. The results are reported in Fig. 28.

Given the same error bound ϵ, the running time of DP using PED is on average (24.3%, 119.9%, 23.4%, 91.3%) of DP using SED. Algorithm DP using PED runs faster than using SED because DP using PED has a better compression ratio than SED, which is the result of less trajectory splitting and distance computing operations during the compression. The running time of algorithm SIPED is on average (7.0%, 36.3%, 19.9%, 69.2%) of algorithm CISED-S on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respec-

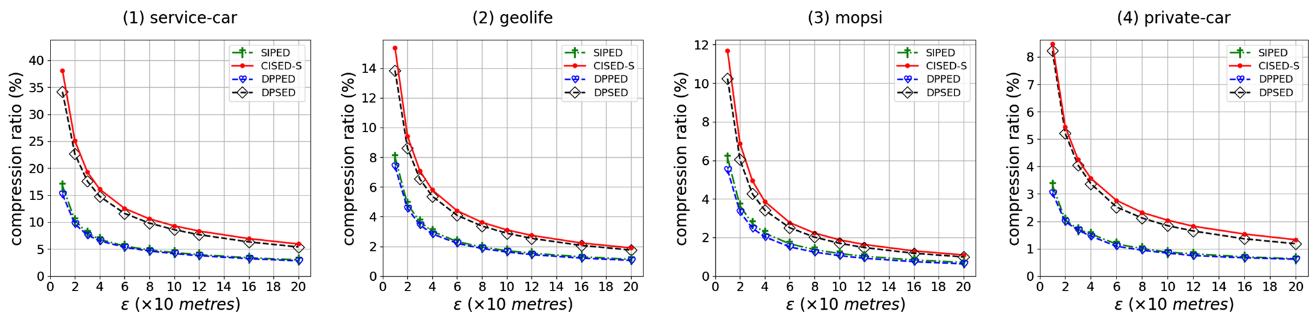

Fig. 26 Evaluation of compression ratios (PED vs. SED): fixed with $m = 16$ and varying error bound ϵ

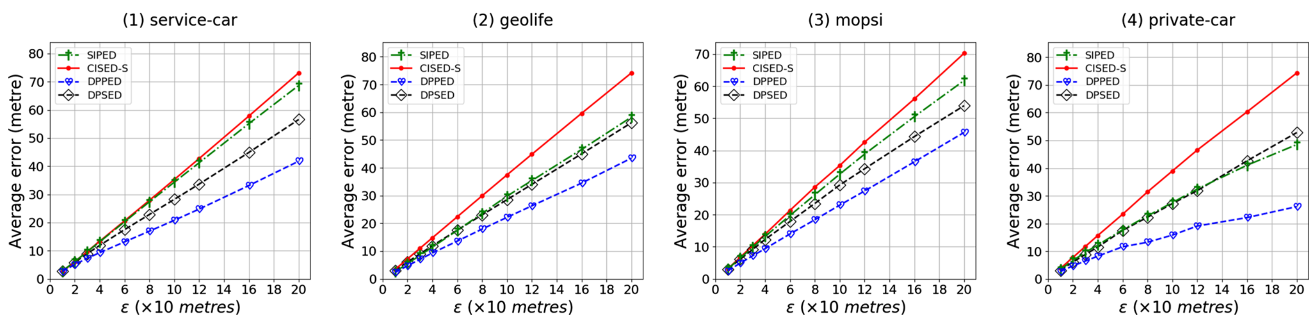

Fig. 27 Evaluation of average errors (PED vs. SED): fixed with $m = 16$ and varying error bound ϵ

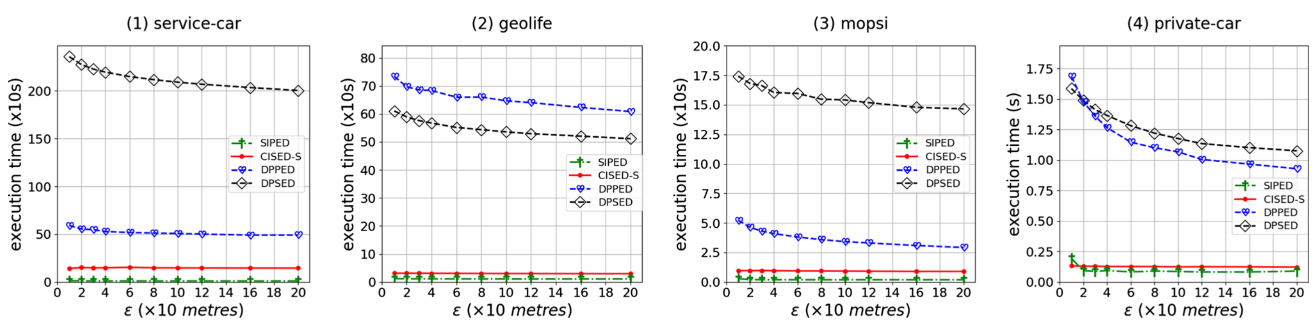

Fig. 28 Evaluation of running time (PED vs. SED): fixed with $m = 16$ and varying error bounds ϵ

tively. Algorithm CISED-S runs slower than SIPED sometimes because finding the common intersection of spatiotemporal cones is a heavier computation than sectors.

5.2.5 Evaluation of spatiotemporal queries on compressed trajectories

In the last set of tests, we evaluate compressed trajectories from the viewpoint of trajectory application, *i.e.,* spatiotemporal query. The well-known spatiotemporal queries are *where_at, when_at, range, nearest_neighbor* and *spatial_join* [1]. Among them, *where_at* query, *i.e.*, "*the position P of a moving object at time t*," is the foundation of *range* and *nearest_neighbor* queries. Hence, we choose it to evaluate compressed trajectories simplified by LS algorithms using PED and/or SED. As mentioned in [1], the answer to *where_at* query is the expected position P' of the moving object at time t. Indeed, it is the *synchronized point* of P when the query is performed on simplified trajectories.

We first compress these trajectories using PED and SED, respectively. When compressing, we also fixed m=16 for algorithms CISED-S and CISED-W, and varied ϵ from 10 to 200 m for all algorithms on the entire four datasets, respectively. Then, for each point P in an original trajectory $\ddot{\mathcal{T}}$, we perform a query on each of its compressed trajectories taking time $P.t$ as input, and calculate the distance between the actual position P and the expected position P' to denote the error of queries. The max and average errors of the queries are reported in Table 3 and Fig. 29, respectively.

(1) When using SED, the max errors of spatiotemporal queries on compressed trajectories are always not larger than error bounds. However, when using PED, they are more than 10^6 meters in datasets ServiceCar, GeoLife and Mopsi, and more than 10^3 meters in dataset PrivateCar, significantly larger

Table 3 The max errors of spatiotemporal queries on compressed trajectories: fixed with $m = 16$ and $\epsilon = 60$ meters

Alg.	ServiceCar	GeoLife	Mopsi	PrivateCar
SIPED	4.81×10^6	1.91×10^6	1.40×10^6	9.45×10^3
DPPED	4.27×10^6	1.03×10^6	4.21×10^6	2.24×10^3
DPSED	59.99	59.99	59.99	59.99
CISED-S	59.99	59.99	59.99	59.99
CISED-W	59.98	59.99	59.99	59.95
SQUISH-E	59.99	59.98	59.99	59.99

than error bounds. This large error phenomenon is actually possible. For example, if someone travels from Beijing of China to Sydney of Australia. During the trip, he first closes his mobile when the air plane is ready to take off, then he turns on his mobile after he arrives at Sydney. Because the sub-trajectory includes data points located in two small regions, *i.e.,* the Beijing and Sydney airports, it is possible that the sub-trajectory is compressed by some LS algorithm using PED into a single line segment, even the error bound ϵ is set to just 60 meters. When a *where_at* is performed on the long line segment, it may return an approximate point having a large error to the actual position. Besides, we also find that the data quality problem of the original given trajectories, such as an abnormal change of GPS location or time, *e.g.,* the latitude and longitude of a moving object suddenly change from one country to another country, or even somewhere in the ocean, can aggravate the large error phenomenon of spatiotemporal queries on compressed trajectories that are simplified using PED. These confirm that SED is more suitable than PED for spatiotemporal queries.

(2) Given the same error bound, the average errors of these queries on compressed trajectories using PED are obvious larger than those using SED, and they are typically larger than error bounds. Moreover, algorithm SI using PED has larger spatiotemporal query errors than DP using PED because SI applies a greedy policy that tries to include as many points into a line segment, and some of these "extra-points" lead to large errors.

5.2.6 Summary and discussion

Summary. From these tests we find the following.

(1) Datasets. The behaviors of all algorithms across all datasets are quite similar.

(2) Polygon intersection algorithms. Algorithm FastRPolyInter runs faster than algorithm CPolyInter and has the same compression ratios and average errors as CPolyInter.

(3) Parameter m. The compression ratio decreases with the increase of m, and the running time increases nearly linearly with the increase of m. In practice, the range of [12, 20] is a good candidate region for m.

(4) Compression ratios. The compression optimal LS algorithm C-Optimal has the best compression ratios among all strong simplification algorithms. Algorithm CISED-S is close to DPSED and algorithm CISED-W is better than all the sub-optimal LS algorithms. The compression ratios of algorithms CISED-S, C-Optimal and CISED-W are on average (79.3%, 71.9%, 67.3%, 72.7%), (58.1%, 45.1%, 39.2%, 52.8%) and (57.7%, 53.8%, 50.0%, 54.6%) of SQUISH-E and (109.2%, 108.0%, 111.7%, 109.1%), (81.3%, 75.5%, 72.5%, 78.1%) and (79.5%, 81.0%, 83.0%, 82.0%) of DPSED on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively.

(5) Average errors. The average errors of these algorithms from the smallest to the largest are SQUISH-E, DPSED, CISED-S, C-Optimal and CISED-W. Algorithm CISED-W has obvious higher average errors than CISED-S as the former essentially forms spatiotemporal cones with a radius of ϵ.

(6) Running time. Algorithms CISED-S and CISED-W are the fastest. They are on average (14.21, 18.19, 17.06, 9.98), (2.84, 3.45, 3.69, 2.86) and (925.25, 7888.26, 40041.59, 8528.76) times faster than DPSED, SQUISH-E and C-Optimalon datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. The advantage of running time of algorithms CISED-S and CISED-W also increases with the increase of the trajectory size.

(7) Distance metrics. Compared with PED, SED supports spatiotemporal queries. However, it comes a price, *e.g.,* the

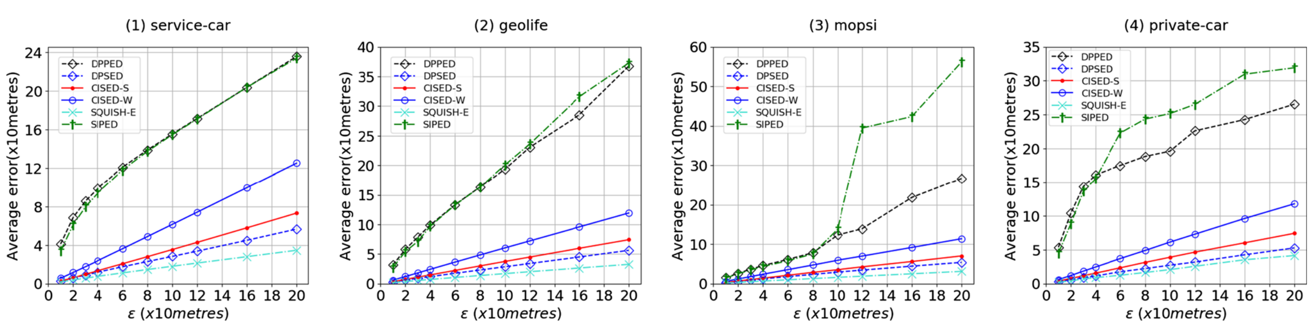

Fig. 29 Average errors of spatiotemporal queries: fixed with $m = 16$ and varying error bound ϵ

compression ratios of algorithms using PED are better than those using SED.

(8) Spatiotemporal Queries. SED is more suitable than PED for applications like spatiotemporal queries.

Discussion. We next briefly discuss the choice of algorithms to compress trajectories. As different applications may have different requirements to reach a balance among multiple metrics, we only provide a brief guideline from the views of running time, compression ratio and average error, respectively.

(1) When the running time is the first-level consideration or algorithms are run in resource-constrained devices, the one-pass algorithms, *i.e.,* CISED-S and CISED-W, are surely the best choices, and they have pretty good compression ratios at the same time.

(2) When the compression ratio is the priority, algorithm CISED-W and the compression optimal algorithm C-Optimal are the selections, followed by algorithms DPSED and CISED-S.

(3) When considering errors, SQUISH-E is a good choice because it has a relative small average error. Alternatively, we can also use DPSED or CISED-S by setting a smaller error bound ϵ compared with SQUISH-E.

6 Related work

Trajectory compression algorithms are normally classified into two categories, namely lossless compression and lossy compression [23]. (1) Lossless compression methods enable exact reconstruction of the original data from the compressed data without information loss. (2) In contrast, lossy compression methods allow errors or deviations, compared with the original trajectories. These techniques typically identify important data points and remove statistical redundant data points from a trajectory, or replace original data points in a trajectory with other places of interests, such as roads and shops. They focus on good compression ratios with acceptable errors. In this work, we focus on lossy compression of trajectory data, and we next introduce the related work on lossy trajectory compression from two aspects: line simplification-based methods and semantics-based methods.

6.1 Line simplification-based methods

The idea of piecewise line simplification comes from computational geometry. Its target is to approximate a given finer piecewise linear curve by another coarser piecewise linear curve, which is typically a subset of the former, such that the maximum distance of the former to the later is bounded by a user specified bound ϵ. Initially, line simplification (LS) algorithms use perpendicular Euclidean distances (PED) as the distance metric. Then, a new distance metric, the synchronous Euclidean distances (SED), was developed after the LS algorithms were introduced to compress trajectories. SED was first introduced in the name of *time-ratio distance* in [19], and formally presented in [29] as the *synchronous Euclidean distance*. PED and SED are two common metrics adopted in trajectory simplification. The former usually brings better compression ratios, while the latter reserves temporal information in the result trajectories. Besides, there is direction based metric [17] that preserves the directions of trajectories.

Line simplification algorithms can be classified into two classes: compression optimal and sub-optimal methods.

6.1.1 Compression optimal algorithms

For the "min-#" problem that finds out the minimal number of points or segments to represent the original polygonal lines *w.r.t.* an error bound ϵ, Imai and Iri [13] first formulated it as a graph problem, and showed that it could be solved in $O(n^3)$ time, where n is the number of the original points. Toussaint of [38] and Melkman and O'Rourke of [18] improved the time complexity to $O(n^2 \log n)$ by using either *convex hull* or *sector intersection* methods. The authors of [2] further proved that the compression optimal algorithm using PED could be implemented in $O(n^2)$ time by using the *sector intersection* mechanism. Because the *sector intersection* and the *convex hull* mechanisms cannot work with SED, hence, currently the time complexity of the compression optimal algorithm using SED remains $O(n^3)$. That is, these compression optimal algorithms are time-consuming and impractical when dealing with large trajectory data [10].

6.1.2 Compression sub-optimal algorithms

Many studies have been targeting at finding the sub-optimal results. In particular, the state-of-the-art of sub-optimal LS approaches fall into three categories, *i.e.,* batch, online and one-pass algorithms. We next introduce these LS-based trajectory compression algorithms from three categories.

Batch algorithms. The batch algorithms adopt a global distance checking policy that requires all trajectory points are loaded before compressing starts. These batch algorithms can be either top-down or bottom-up.

Top-down algorithms, *e.g.,* Ramer [30] and Douglas–Peucker [6], recursively divide a trajectory into sub-trajectories until the stopping condition is met. Bottom-up algorithms, *e.g.,* Theo Pavlidis' algorithm [26], is the natural complement of the top-down ones, which 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. The distances of newly generated line segments are recalculated during the process. These algorithms originally only support PED, but are easy to be extended to support SED [19]. The batch nature and high time complexities make batch algorithms impractical for online and/or resource-constrained scenarios [14].

Online algorithms. The online algorithms adopt a constrained global distance checking policy that restricts the checking within a sliding or opening window. Constrained global checking algorithms do not need to have the entire trajectory ready before they start compressing, and are more appropriate than batch algorithm for compressing trajectories for online scenarios.

Several LS algorithms have been developed, *e.g.,* by combining DP or Theo Pavlidis' with sliding or opening windows for online processing [19]. These methods still have a high time and/or space complexity, which significantly hinders their utility in resource-constrained mobile devices [15]. BQS [15,16] and SQUISH-E [23] further optimize the opening window algorithms. BQS [15,16] speeds up the processing by picking out at most eight special points from an open window based on a convex hull, which, however, hardly supports SED. The SQUISH-E [23] algorithm is a combination of opening window and bottom-up online algorithm. It uses a doubly linked list Q to achieve a better efficiency. Although SQUISH-E supports SED, it is not one-pass, and has a relatively poor compression ratio.

One-pass algorithms. The one-pass algorithms adopt a local distance checking policy. They do not need a window to buffer the previously read points as they process each point in a trajectory once and only once. Obviously, the one-pass algorithms run in linear time and constant space.

The n–th point routine and the routine of random-selection of points [35] are two naive one-pass algorithms. In these routines, for every fixed number of consecutive points along the line, the n–th point and one random point among them are retained, respectively. They are fast, but are obviously not error bounded. In Reumann–Witkam routine [31], it builds a strip paralleling to the line connecting the first two points; then, the points within this strip compose one section of the line. The Reumann–Witkam routine also runs fast, but has limited compression ratios. The sector intersection (SI) algorithm [36,40] was developed for graphic and pattern recognition in the late 1970s, for the approximation of arbitrary planar curves by linear segments or finding a polygonal approximation of a set of input data points in a 2D Cartesian coordinate system. [7] optimized algorithm SI by considering the distance between a potential end point and the initial point of a line segment, and the Sleeve algorithm [42] in the cartographic discipline essentially applies the same idea as the SI algorithm. The authors of this article also developed a one-pass error bounded (OPERB) algorithm [14]. However, all existing one-pass algorithms use PED [7,14,36,40,42], while this study focuses on SED.

6.2 Semantics-based methods

The trajectories of certain moving objects such as cars and trucks are constrained by road networks. These moving objects typically travel along road networks, instead of the line segment between two points. Trajectory compression methods based on road networks [3,5,8,9,12,28,37] project trajectory points onto roads (also known as Map-Matching). Moreover, [8,9,37] mine and use high frequency patterns of compressed trajectories, instead of roads, to further improve compression effectiveness. Some methods [32,33] compress trajectories beyond the use of road networks and further make use of other user specified domain knowledge, such as places of interests along the trajectories [32].

These semantics-based approaches are orthogonal to line simplification-based methods and may be combined with each other to improve the effectiveness of trajectory compression.

7 Conclusions

We have proposed CISED-S and CISED-W, two one-pass error bounded strong and weak trajectory simplification algorithms using the synchronous distance. We have also experimentally verified that algorithms CISED-S and CISED-W are fast and have good compression ratios. They are three times faster than SQUISH-E, the fastest existing LS algorithm using SED. In terms of compression ratio, algorithm CISED-S is close to DPSED, the existing LS algorithm with the best compression ratio, and is 21.1% better than SQUISH-E on average; and algorithm CISED-W is better than all the sub-optimal algorithm and is on average 19.6% and 42.4% better than DPSED and SQUISH-E, respectively.

Acknowledgements This work is supported in part by National Key R&D Program of China 2018YFB1700403, NSFC U1636210 &61421003, Shenzhen Institute of Computing Sciences, and the Fundamental Research Funds for the Central Universities. Funding was provided by National Natural Science Foundation of China (Grant No. U1636210), 973 program (Grant No. 2014CB340300), Beijing Advanced Innovation Center for Big Data and Brain Computing.

Appendix: Proofs

Proof of Proposition 1 Let P'_{s+i} $(i \in [1, k])$ be the intersection point of line segment $\overrightarrow{P_s Q}$ and the plane $P.t - P_{s+i}.t = 0$. It suffices to show that the intersection point P'_{s+i} satisfies that (1) $P'_{s+i}.t = P_{s+i}.t$, (2) $P'_{s+i}.x = P_s.x + w \cdot (Q.x - P_s.x)$, and (3) $P'_{s+i}.y = P_s.y + w \cdot (Q.y - P_s.y)$, where $w = \frac{P'_{s+i}.t - P_s.t}{Q.t - P_s.t} = \frac{P_{s+i}.t - P_s.t}{Q.t - P_s.t}$. Hence, P'_{s+i} is indeed a synchronized point of P_{s+i} *w.r.t.* $\overrightarrow{P_s Q}$, and the distance $|\overrightarrow{P_{s+i} P'_{s+i}}|$

from P_{s+i} to P'_{s+i} is the synchronous distance of P_{s+i} to $\overrightarrow{P_s Q}$.

We assume first that $\bigsqcap_{i=1}^{k} \mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon)) \neq \{P_s\}$. Then, there must exist a point Q in the area of the synchronous circle $\mathcal{O}(P_{s+k}, \epsilon)$ such that $\overrightarrow{P_s Q}$ passes through all the cones $\mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon))$ $i \in [1, k]$. Hence, $Q.t = P_{s+k}.t$. We also have $sed(P_{s+i}, \overrightarrow{P_s Q}) = |\overrightarrow{P'_{s+i} P_{s+i}}| \leq \epsilon$ for each $i \in [1, k]$ since P'_{s+i} is in the area of circle $\mathcal{O}(P_{s+i}, \epsilon)$.

Conversely, assume that there exists a point Q such that $Q.t = P_{s+k}.t$ and $sed(P_{s+i}, \overrightarrow{P_s Q}) \leq \epsilon$ for all P_{s+i} ($i \in [1, k]$). Then, $|\overrightarrow{P'_{s+i} P_{s+i}}| \leq \epsilon$ for all $i \in [1, k]$. Hence, we have $\bigsqcap_{i=1}^{k} \mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon)) \neq \{P_s\}$. □

Proof of Proposition 2 By Proposition 1, it suffices to show that $\bigsqcap_{i=1}^{k} \mathcal{O}^c(P^c_{s+i}, r^c_{s+i}) \neq \emptyset$ if and only if $\bigsqcap_{i=1}^{k} \mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon)) \neq \{P_s\}$, which is obvious. Hence, we have the conclusion. □

Proof of Proposition 3 We shall prove this by contradiction. Assume first that $\mathcal{R}^*_l \bigsqcap \mathcal{R}_{s+l+1}$ has more than m edges. Then, it must have two distinct edges $\overrightarrow{A_i}$ and $\overrightarrow{A_{i'}}$ with the same label j ($1 \leq j \leq m$), originally from $\mathcal{R}_{s+i}$ and $\mathcal{R}_{s+i'}$ ($1 \leq i < i' \leq l+1$). Note that here $\mathcal{R}_{s+i} \bigsqcap \mathcal{R}_{s+i'} \neq \emptyset$ since $\mathcal{R}^*_l \bigsqcap \mathcal{R}_{s+l+1} \neq \emptyset$. However, when $\mathcal{R}_{s+i} \bigsqcap \mathcal{R}_{s+i'} \neq \emptyset$, the intersection $\mathcal{R}_{s+i} \bigsqcap \mathcal{R}_{s+i'}$ cannot have both edge $\overrightarrow{A_i}$ and edge $\overrightarrow{A_{i'}}$, as all edges with the same label are in parallel (or overlapping) with each other by the above definition of inscribed regular polygons. This contradicts with the assumption. □

Proof of Proposition 4 The inscribed regular polygon $\mathcal{R}_{s+l+1}$ has m edges, and intersection polygon $\mathcal{R}^*_l$ has at most m edges by Proposition 3. As the intersection of two m-edges convex polygons can be computed in $O(m)$ time [25], the intersection of polygons $\mathcal{R}^*_l$ and $\mathcal{R}_{s+l+1}$ can be done in $O(1)$ time for a fixed m. □

Proof of Proposition 5 If ($\overrightarrow{A} \bigsqcap \overrightarrow{B} \neq \emptyset$ **and** $\overrightarrow{A} \times \overrightarrow{B} < 0$ **and** $P_{e_A} \notin \mathcal{H}(\overrightarrow{B})$) or ($\overrightarrow{A} \bigsqcap \overrightarrow{B} \neq \emptyset$ **and** $\overrightarrow{A} \times \overrightarrow{B} \geq 0$ **and** $P_{e_B} \in \mathcal{H}(\overrightarrow{A})$), then as all edges in the same edge group E^j ($1 \leq j \leq m$) are in parallel with each other and by the geometric properties of regular polygon $\mathcal{R}_{s+k+1}$, it is easy to find that, for each position of $\overrightarrow{A}$ between its original to its opposite positions, we have (1) $\overrightarrow{A} \bigsqcap \overrightarrow{B} = \emptyset$, and (2) either $P_{e_A} \notin \mathcal{H}(\overrightarrow{B})$ or $P_{e_B} \in \mathcal{H}(\overrightarrow{A})$. Hence, by the advance rule (1) of algorithm CPolyInter in Sect. 2.4, edge $\overrightarrow{A}$ is always moved forward until it reaches the opposite position of its original one. From this, we have the conclusion. □

Proof of Proposition 6 If ($\overrightarrow{A} \bigsqcap \overrightarrow{B} \neq \emptyset$ **and** $\overrightarrow{A} \times \overrightarrow{B} \geq 0$ **and** $P_{e_B} \notin \mathcal{H}(\overrightarrow{A})$) or ($\overrightarrow{A} \bigsqcap \overrightarrow{B} \neq \emptyset$ **and** $\overrightarrow{A} \times \overrightarrow{B} < 0$ **and** $P_{e_A} \in \mathcal{H}(\overrightarrow{B})$), then it is also easy to find that, for each position of $\overrightarrow{B}$ between its original to its target positions (*i.e.,* the edge after the one having the same edge group as $\overrightarrow{A}$), we have (1) $\overrightarrow{A} \bigsqcap \overrightarrow{B} = \emptyset$, and (2) either $P_{e_B} \notin \mathcal{H}(\overrightarrow{A})$ or $P_{e_A} \in \mathcal{H}(\overrightarrow{B})$. Hence, by the advance rule (2) of algorithm CPolyInter in Sect. 2.4, edge $\overrightarrow{B}$ is always moved forward until it reaches the target position. From this, we have the conclusion. □

Proof of Proposition 8 If $\bigsqcap_{i=s+1}^{e} \mathcal{C}(P_s, P_{s+i}, \epsilon/2) \neq \{P_s\}$, then by Proposition 1, there exists a point Q, $Q.t = P_{s+k}.t$, such that $sed(P_{s+i}, \overrightarrow{P_s Q}) \leq \epsilon/2$ for all $i \in [1, k]$. By the triangle inequality essentially, $sed(P_{s+i}, \overrightarrow{P_s P_{s+k}}) \leq sed(P_{s+i}, \overrightarrow{P_s Q}) + |\overrightarrow{Q P_{s+k}}| \leq \epsilon/2 + \epsilon/2 = \epsilon$. □

Proof of Proposition 9 By Proposition 2 and the nature of inscribed regular polygon, it is easy to find that for any point $Q \in \mathcal{R}^*_k$ *w.r.t.* plane $t_c = P_{s+k}.t$, there is $sed(P_{s+i}, \overrightarrow{P_s Q}) \leq \epsilon$ for all points P_{s+i} ($i \in [1, k]$). From this, we have the conclusion. □

References

1. Cao, H., Wolfson, O., Trajcevski, G.: Spatio-temporal data reduction with deterministic error bounds. VLDBJ **15**(3), 211–228 (2006)
2. Chan, W., Chin, F.: Approximation of polygonal curves with minimum number of line segments or minimal error. Int. J. Comput. Geom. Appl. **6**(1), 378–387 (1996)
3. Chen, Y., Jiang, K., Zheng, Y., Li, C., Yu, N.: Trajectory simplification method for location-based social networking services. In: LBSN, pp. 33–40 (2009)
4. Chen, M., Xu, M., Fränti, P.: A fast multiresolution polygonal approximation algorithm for GPS trajectory simplification. IEEE Trans. Image Process. **21**(5), 2770–2785 (2012)
5. Civilis, A., Jensen, C.S., Pakalnis, S.: Techniques for efficient road-network-based tracking of moving objects. TKDE **17**(5), 698–712 (2005)
6. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Can. Cartogr. **10**(2), 112–122 (1973)
7. Dunham, J.G.: Piecewise linear approximation of planar curves. PAMI **8**, 9–67 (1986)
8. Gotsman, R., Kanza, Y.: A dilution-matching-encoding compaction of trajectories over road networks. GeoInformatica **19**(2), 331–3364 (2015)
9. Han, Y., Sun, W., Zheng, B.: Compress: a comprehensive framework of trajectory compression in road networks. TODS **42**(2), 11:1–11:9 (2017)
10. Heckbert, P.S., Garland, M.: Survey of polygonal surface simplification algorithms. In: SIGGRAPH (1997)
11. Hershberger, J., Snoeyink, J.: Speeding up the Douglas–Peucker line-simplification algorithm. Technical Report, University of British Columbia (1992)
12. Hung, C.C., Peng, W., Lee, W.: Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. VLDBJ **24**(2), 169–192 (2015)
13. Imai, H., Iri, M.: Computational-geometric methods for polygonal approximations of a curve. Comput. Vis. Graph. Image Process. **36**, 31–41 (1986)

14. Lin, X., Ma, S., Zhang, H., Wo, T., Huai, J.: One-pass error bounded trajectory simplification. PVLDB **10**(7), 841–852 (2017)
15. Liu, J., Zhao, K., Sommer, P., Shang, S., Kusy, B., Jurdak, R.: Bounded quadrant system: Error-bounded trajectory compression on the go. In: ICDE (2015)
16. Liu, J., Zhao, K., Sommer, P., Shang, S., Kusy, B., Lee, J.-G., Jurdak, R.: A novel framework for online amnesic trajectory compression in resource-constrained environments. IEEE Trans. Knowl. Data Eng. **28**(11), 2827–2841 (2016)
17. Long, C., Wong, R.C.-W., Jagadish, H.: Direction-preserving trajectory simplification. PVLDB **6**(10), 949–960 (2013)
18. Melkman, A., O'Rourke, J.: On polygonal chain approximation. Mach. Intell. Pattern Recognit. **6**, 87–95 (1988)
19. Meratnia, N., de By, R.A.: Spatiotemporal compression techniques for moving point objects. In: EDBT (2004)
20. Metha, R., Mehta, V.K.: The Principles of Physics. S Chand, New Delhi (1999)
21. Mopsi routes 2014. http://cs.uef.fi/mopsi/routes/dataset/. Accessed 29 Nov 2017
22. Muckell, J., Hwang, J.-H., Lawson, C.T., Ravi, S.S.: Algorithms for compressing gps trajectory data: an empirical evaluation. In: ACM-GIS (2010)
23. Muckell, J., Olsen, P.W., Hwang, J.-H., Lawson, C.T., Ravi, S.S.: Compression of trajectory data: a comprehensive evaluation and new approach. GeoInformatica **18**(3), 435–460 (2014)
24. Nibali, A., He, Z.: Trajic: an effective compression system for trajectory data. TKDE **27**(11), 3138–3151 (2015)
25. O'Rourke, J., Chien, C.B., Olson, T., Naddor, D.: A new linear algorithm for intersecting convex polygons. Comput. Graph. Image Process. **19**(4), 384–391 (1982)
26. Pavlidis, T., Horowitz, S.L.: Segmentation of plane curves. IEEE Trans. Comput. **23**(8), 860–870 (1974)
27. Pfoser, D., Jensen, C.S.: Capturing the uncertainty of moving-object representations. In: SSD (1999)
28. Popa, I.S., Zeitouni, K., Oria, V., Kharrat, A.: Spatio-temporal compression of trajectories in road networks. GeoInformatica **19**(1), 117–145 (2014)
29. Potamias, M., Patroumpas, K., Sellis, T.K.: Sampling trajectory streams with spatiotemporal criteria. In: SSDBM (2006)
30. Ramer, U.: An iterative procedure for the polygonal approximation of plane curves. Comput. Graph. Image Process. **1**, 244–256 (1972)
31. Reumann, K., Witkam, A.: Optimizing curve segmentation in computer graphics. In: International Computing Symposium (1974)
32. Richter, K.-F., Schmid, F., Laube, P.: Semantic trajectory compression: representing urban movement in a nutshell. J. Spat. Inf. Sci. **4**(1), 3–30 (2012)
33. Schmid, F., Richter, K., Laube, P.: Semantic trajectory compression. In: SSTD, pp. 411–416 (2009)
34. Shamos, M.I., Dan, H.: Geometric intersection problems. In: Symposium on Foundations of Computer Science, pp. 208–215 (1976)
35. Shi, W., Cheung, C.: Performance evaluation of line simplification algorithms for vector generalization. Cartogr. J. **43**(1), 27–44 (2006)
36. Sklansky, J., Gonzalez, V.: Fast polygonal approximation of digitized curves. Pattern Recognit. **12**, 327–331 (1980)
37. Song, R., Sun, W., Zheng, B., Zheng, Y.: Press: a novel framework of trajectory compression in road networks. PVLDB **7**(9), 661–672 (2014)
38. Toussaint, G.T.: On the complexity of approximating polygonal curves in the plane. In: International Symposium on Robotics and Automation (IASTED) (1985)
39. Trajcevski, G., Cao, H., Scheuermanny, P., Wolfsonz, O., Vaccaro, D.: On-line data reduction and the quality of history in moving objects databases. In: MobiDE (2006)
40. Williams, C.M.: An efficient algorithm for the piecewise linear approximation of planar curves. Comput. Graph. Image Process. **8**, 286–293 (1978)
41. Zhang, D., Ding, M., Yang, D., Liu, Y., Fan, J., Shen, H.T.: Trajectory simplification: an experimental study and quality analysis. PVLDB **9**(11), 934–946 (2018)
42. Zhao, Z., Saalfeld, A.: Linear-time sleeve-fitting polyline simplification algorithms. In: Proceedings of AutoCarto, pp. 214–223 (1997)
43. Zheng, Y., Xie, X., Ma, W.: GeoLife: a collaborative social networking service among user, location and trajectory. IEEE Data Eng. Bull. **33**(2), 32–39 (2010)
44. Züfle, A., Trajcevski, G., Pfoser, D., Renz, M., Rice, M.T., Leslie, T., Delamater, P.L., Emrich, T.: Handling uncertainty in geo-spatial data. In: ICDE (2017)