Noname manuscript No.
(will be inserted by the editor)

One-Pass Trajectory Simplification Using the Synchronous Euclidean Distance

Xuelian Lin · Jiahao Jiang · Shuai Ma · Yimeng Zuo · Chunming Hu

Received: xxx, 2017 / Accepted: xxx, 2018

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 band and computing resource. To attack this issue, one-pass line simplification (LS) algorithms have are 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 spatio-temporal 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 spatio-temporal 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 most efficient existing LS algorithm using SED). In terms of compression ratios, algorithms CISED-S and CISED-W are comparable with and 19.6% better than DPSED (the most effective existing LS algorithm using SED) on average, respectively, and are 21.1% and 42.4% better than SQUISH-E on average, respectively.

1 Introduction

Various mobile devices, such as smart-phones, on-board diagnostics, personal navigation devices, and wearable smart devices, have been using their sensors to collect massive trajectory data of moving objects at a certain sampling rate (e.g., a data point every 5 seconds), which is 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 [2, 5, 15–17, 20, 22–24, 27, 34]. It is known that these issues can be resolved or greatly alleviated by trajectory compression techniques via removing redundant data points of trajectories [2,4,5,7,10,12,15–18,20,23,24,27], among which the piece-wise line simplification technique is widely used [2, 4, 5, 7, 15–17, 20, 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 [15, 27].

Originally, line simplification (LS) algorithms adopt the *perpendicular Euclidean distance* (PED) as a metric to compute the errors, *e.g.,* $|\overrightarrow{P_4P_4^*}|$ is the PED of data point P_4 to line segment $\overrightarrow{P_0P_{10}}$ in Figure 1 (left). Line simplification algorithms using PED have good compression ratios [2, 5, 7, 12, 15, 16, 23, 34]. However, when using PED, the temporal information is lost. Thus, a spatio-temporal query, *e.g.,* "*the position of a moving object at time t*", on the compressed trajectories by LS algorithms using PED may return an approximate point P' whose distance to the actual position P of the moving object at time t is unbounded.

The *synchronous Euclidean distance* (SED) was then introduced for trajectory compression to support the above spatio-temporal queries [20]. SED is the Euclidean distance of a data point to its *approximate temporally synchronized data point* [20] on the corresponding line segment. For instance, P_4' and P_7' are the *synchronized data points* of points P_4 and P_7 *w.r.t.* line segments $\overrightarrow{P_0P_{10}}$ and $\overrightarrow{P_4P_{10}}$, respectively, in Figure 1 (right). LS algorithms using SED may produce more

X. Lin, J. Jiang, S. Ma (correspondence), Y. Zuo and C. Hu
SKLSDE lab, School of Computer Science and Engineering, Beihang University, China.
E-mail: {linxl, jiangjh, mashuai, zuoym, hucm}@buaa.edu.cn

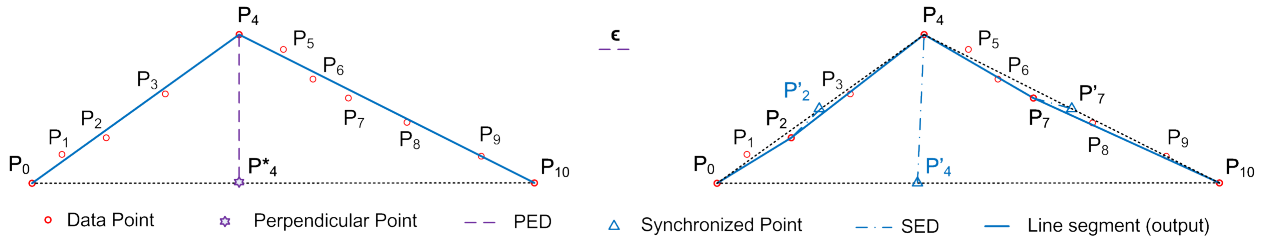

Fig. 1 A trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ with eleven points is represented by two (left) and four (right) continuous line segments (solid blue), compressed by the Douglas–Peucker algorithm [7] using PED and SED, respectively. The Douglas–Peucker algorithm firstly creates line segment $\overrightarrow{P_0P_{10}}$, 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.

line segments. However, the use of SED ensures that the Euclidean distance between a data point and its synchronized point *w.r.t.* the corresponding line segment is limited within a distance bound ϵ. Hence, the above spatio-temporal query over the trajectories compressed by SED enabled approaches returns the synchronized point P' of a data point P within the bound ϵ.

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 [3, 14]. An optimal $O(n^3)$ LS algorithm using PED was firstly developed in [14], where n is the number of the original points. Later, an improved optimal $O(n^2)$ algorithm using PED was designed in [3], with the help of *sector intersection* mechanism. However, the time complexity of the optimal LS algorithm using SED remains in $O(n^3)$, as the optimization mechanisms are PED specific, and cannot work with SED.

Due to the high time complexities of optimal LS algorithms using SED, sub-optimal LS algorithms using SED have been developed for trajectory compression, including batch algorithms (*e.g.,* Douglas-Peucker based algorithm DPSED [20]) and online algorithms (*e.g.,* SQUISH-E [23]). However, these methods still have high time and/or space complexities, which hinder their utilities in resource-constrained devices.

Observe that linear time LS algorithms using PED [8, 15, 35, 39, 40] have been develped, and they are more efficient for resource-constrained devices. The key idea to achieve a linear time complexity is by local distance checking in constant time, *e.g.,* the *sector intersection* mechanism used in [8, 35, 39, 40] and the *fitting function* approach used in our preview work [15]. Unfortunately, these techinques are designed specifically for PED, and cannot be applied for SED.

Indeed, it is even more challenging to design an one-pass LS algorithm using SED than using PED. To our knowledge, no one-pass LS algorithms using SED have been develoed in the community.

Contributions. To this end, we propose two one-pass error bounded LS algorithms using SED for compressing trajectories in an efficient and effective way.

(1) We first develop a novel local synchronous distance checking approach, *i.e.,* spatio-temporal Cone Intersection using the Synchronous Euclidean Distance (CISED). We further approximate the intersection of spatio-temporal cones with the intersection of regular polygons, and develop a fast regular polygon intersection algorithm, such that each data point in a trajectory is checked in $O(1)$ time during the entire process of trajectory simplification.

(2) We next develop 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 output.

(3) Using four real-life trajectory datasets (Truck, ServiceCar, GeoLife, PrivateCar), we finally conduct an extensive experimental study, by comparing our methods CISED-S and CISED-W with the optimal LS algorithm using SED, DPSED [20] (the most effective existing LS algorithm using SED) and SQUISH-E [23] (the most efficient existing LS algorithm using SED).

For running time, algorithms CISED-S and CISED-W 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 the optimal LS algorithm on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively.

For compression ratios, algorithm CISED-S is better than SQUISH-E and comparable with DPSED. The output sizes of CISED-S are on average (79.5%, 79.5%, 66.0%, 72.7%), (109.1%, 109.7%, 113.5%, 109.2%) and (134.3%, 133.3%, 148.4%, 135.4%) of SQUISH-E, DPSED and the optimal LS algorithm on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively.

Moreover, algorithm CISED-W is comparable with the optimal LS algorithm and better than SQUISH-E and DPSED that are on average (57.9%, 58.8%, 48.4%, 54.6%), (79.6%, 81.2%, 83.3%, 82.1%) and (98.1%, 98.7%, 108.9%, 101.7%) of SQUISH-E, DPSED and the optimal LS algorithm on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively.

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 Section 6 and conclusion in Section 7.

2 Preliminaries

In this section, we first introduce basic concepts for piece-wise line based trajectory compression. We then describe the optimal LS algorithm and the *sector intersection* mechanism, and show how this mechanism can be used to fast 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.

Notations used are summarized in Table 1.

Table 1 Summary of notations

Notations	Semantics
P	a data point
$\ddot{\mathcal{T}}$	a trajectory $\ddot{\mathcal{T}}$ is a sequence of data points
$\overline{\mathcal{T}}$	a piece-wise line representation of a trajectory $\ddot{\mathcal{T}}$
$\mathcal{L}$	a directed line segment
$ped(P, \mathcal{L})$	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
$\vec{A} \times \vec{B}$	the cross product of (vectors) $\vec{A}$ and $\vec{B}$
$\mathcal{H}(\mathcal{L})$	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 spatio-temporal cone
$\mathcal{O}^c$	a cone projection circle
$\bigcap$	intersection of geometries
G	the reachability graph of a trajectory

2.1 Basic Notations

We first introduce 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 a three-dimension 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 $0 \leq i < j \leq n$). Intuitively, a trajectory is the path (or track) that a moving object follows through space as a function of time [21].

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

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

Piecewise line representation ($\overline{\mathcal{T}}$). A piece-wise line representation $\overline{\mathcal{T}}[\mathcal{L}_0, \ldots, \mathcal{L}_m]$ $(0 < m \leq n)$ of a trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ is a sequence of continuous directed line segments $\mathcal{L}_i = \overrightarrow{P_{s_i}P_{e_i}}$ $(i \in [0, m])$ 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 $\mathcal{L} = \overrightarrow{P_sP_e}$, the perpendicular Euclidean distance (or simply perpendicular distance) $ped(P, \mathcal{L})$ of point P to line segment $\mathcal{L}$ is $\min\{|PQ|\}$ for any point Q on $\overrightarrow{P_sP_e}$.

Synchronized points [20]. 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 $\overrightarrow{P_sP_e}$ is defined as follows: (1) $P'.x = P_s.x + c \cdot (P_e.x - P_s.x)$, (2) $P'.y = P_s.y + c \cdot (P_e.y - P_s.y)$ and (3) $P'.t = P.t$, where $c = \frac{P.t - P_s.t}{P_e.t - P_s.t}$.

Synchronous Euclidean Distance (SED) [20]. Given a data point P and a directed line segment $\mathcal{L}$

$= \overrightarrow{P_sP_e}$, 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 data point P' *w.r.t.* $\mathcal{L}$.

We illustrate these notations with examples.

Example 1 Consider Figure 1, in which
(1) $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ is a trajectory having 11 data points,
(2) the set of two continuous line segments $\{\overrightarrow{P_0P_4}, \overrightarrow{P_4P_{10}}\}$ (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 $\ddot{\mathcal{T}}$,
(3) $ped(P_4, \overrightarrow{P_0P_{10}}) = |\overrightarrow{P_4P_4^*}|$, where P_4^* is the perpendicular point of P_4 *w.r.t.* line segment $\overrightarrow{P_0P_{10}}$, and
(4) $sed(P_4, \overrightarrow{P_0P_{10}}) = |\overrightarrow{P_4P_4'}|$, $sed(P_2, \overrightarrow{P_0P_4}) = |\overrightarrow{P_2P_2'}|$ 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 $\ddot{\mathcal{T}}$ and its compression algorithm $\mathcal{A}$ using SED (respectively PED) that produces another trajectory $\ddot{\mathcal{T}}'$, we say that algorithm $\mathcal{A}$ is error bounded by ϵ if for each point P in $\ddot{\mathcal{T}}$, there exist points P_j and P_{j+1} in $\ddot{\mathcal{T}}'$ such that $sed(P, \mathcal{L}(P_j, P_{j+1})) \leq \epsilon$ (respectively $ped(P, \mathcal{L}(P_j, P_{j+1})) \leq \epsilon$).

2.2 The Optimal LS Algorithm

Given a trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ and an error bound ϵ, the optimal trajectory simplification problem, as formulated by Imai and Iri in [14], can be solved in two steps: (1) construct a reachability graph G of $\ddot{\mathcal{T}}$ and (2) search a shortest path from P_0 to P_n in graph G.

The reachability graph of a trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$ *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 $P_{s+k} \in V$ $(s \geq 0, k > 0, s+k \leq n)$, edge $(P_s, P_{s+k}) \in E$ if and only if the distance of each point P_{s+i} $(i \in [0, k])$ to line segment $\overrightarrow{P_sP_{s+k}}$ is no 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 $\ddot{\mathcal{T}}$. The path also reveals the subset of points of $\ddot{\mathcal{T}}$ used in the approximate trajectory, (2) the path length corresponds to the number of line segments in the approximation trajectory, and (3) a shortest path is an optimal representation of trajectory $\ddot{\mathcal{T}}$.

Constructing the reachability graph G needs to check for all pair of points P_s and P_{s+k} whether the distances of all points (P_{s+i}, $0 < i < k$) to the line segment $\overrightarrow{P_sP_{s+k}}$ 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 [3], the construction of the reachability graph G using PED can be implemented in $O(n^2)$ time using the *sector intersection* mechanism (see Section 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 [35, 39] 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 [40] in the cartographic discipline essentially applies the same idea as the SI algorithm. Further, [8] 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 distances.

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 $\overrightarrow{P_sP_{s+i}^u}$ and $\overrightarrow{P_sP_{s+i}^l}$ such that $ped(P_{s+i}, \overrightarrow{P_sP_{s+i}^u}) = ped(P_{s+i}, \overrightarrow{P_sP_{s+i}^l}) = \epsilon$ and either ($\overrightarrow{P_sP_{s+i}^l}.\theta < \overrightarrow{P_sP_{s+i}^u}.\theta$ *and* $\overrightarrow{P_sP_{s+i}^u}.\theta - \overrightarrow{P_sP_{s+i}^l}.\theta < \pi$) or ($\overrightarrow{P_sP_{s+i}^l}.\theta > \overrightarrow{P_sP_{s+i}^u}.\theta$ *and* $\overrightarrow{P_sP_{s+i}^u}.\theta - \overrightarrow{P_sP_{s+i}^l}.\theta < -\pi$). Indeed, they forms a *sector* $\mathcal{S}(P_s, P_{s+i}, \epsilon)$ that takes P_s as the center point and $\overrightarrow{P_sP_{s+i}^u}$ and $\overrightarrow{P_sP_{s+i}^l}$ as the 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 no greater than the error bound ϵ if and only if the k sectors $\mathcal{S}(P_s, P_{s+i}, \epsilon)$ $(i \in [1, k])$ share common data points other than P_s, *i.e.*, $\bigcap_{i=1}^{k} \mathcal{S}(P_s, P_{s+i}, \epsilon) \neq \{P_s\}$ [35, 39, 40].

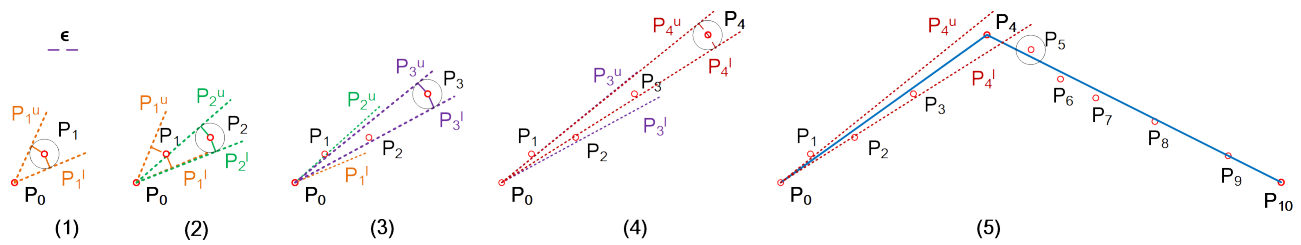

Fig. 2 Trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ in Figure 1 is compressed into two line segments by the Sector Intersection algorithm [35,39].

The point Q may not belong to $\{P_s, P_{s+1}, \ldots, P_{s+k}\}$. However, if P_{s+i} $(1 \leq i \leq k)$ is chosen as Q, then for any data point P_{s+j} $(j \in [1, ...i])$, its perpendicular Euclidean distance to line segment $\overline{P_sP_{s+i}}$ is no greater than the error bound ϵ if and only if $\bigcap_{j=1}^{i} \mathcal{S}(P_s, P_{s+j}, \epsilon/2) \neq \{P_s\}$, as pointed out in [40].

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.

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} $(i \in [1, ...k])$, its synchronized Euclidean distance to directed line $\overrightarrow{P_sQ}$ is no greater than the error bound ϵ". Hence, the SI mechanism is PED specific, and not applicable for SED.

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

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

(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 Figure 2.(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 $P_0P_2^u$ and a low border line $P_0P_1^l$, as shown in Figure 2.(2).

Similarly, points P_3 and P_4 are processed, as shown in Figures 2.(3) and 2.(4), respectively.

(3) When point P_5 is read, line segment $\overrightarrow{P_0P_4}$ is produced, and point P_4 becomes the start point, as $\bigcap_{i=1}^{4} \mathcal{S}(P_0, P_{s+i}, \epsilon/2) \neq \{P_0\}$ and $\bigcap_{i=1}^{5} \mathcal{S}(P_0, P_{s+i}, \epsilon/2) = \{P_0\}$ as shown in Figure 2.(5).

(4) Points $P_5, \ldots, P_{10}$ are processed similarly one by one in order, and finally the algorithm outputs another line segment $\overrightarrow{P_4P_{10}}$ as shown in Figure 2.(5). □

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 $\mathcal{R}_1$, repectively (see Figure 4).

The algorithm has $\vec{A}$ and $\vec{B}$ "chasing" one another, *i.e.,* moves $\vec{A}$ on $\mathcal{R}_2$ and $\vec{B}$ on $\mathcal{R}_1$ counter-clockwise 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 $\mathcal{H}(\vec{A})$ 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 Figure 4.(1) and 4.(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 Figure 4.(6) and 4.(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 Figure 3. 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 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 $\vec{B}$ 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

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

Fig. 3 Algorithm for convex polygons intersection [25].

$\mathcal{R}_1$, and $\mathcal{R}_1 \bigcap \mathcal{R}_2 = \emptyset$ (line 13). At last, it returns the intersection polygon (line 14).

The algorithm 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 worth pointing out that $|\mathcal{R}_1 \bigcap \mathcal{R}_2| \leq (|\mathcal{R}_1| + |\mathcal{R}_2|)$.

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

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

(2) Then, by advance rule (1), edge $\overrightarrow{A}$ moves on a step and makes $\overrightarrow{A} \bigcap \overrightarrow{B} = \emptyset$ as shown in Figure 4.(2). After 7 steps of moving of edge $\overrightarrow{A}$ or $\overrightarrow{B}$, each by an advance rule, $\overrightarrow{A}$ and $\overrightarrow{B}$ intersect on P_2, as shown in Figure 4.(6).

(3) Next, edge $\overrightarrow{B}$ moves on a step, and makes $\overrightarrow{A} \bigcap \overrightarrow{B} = \emptyset$, as shown in Figure 4.(7).

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

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

3 Local Synchronous Distance Checking

In this section, we develop a local synchronous distance checking approach such that each point in a trajectory is checked only once in $O(1)$ time during the entire process of trajectory simplification, by substantially extending the *sector intersection* method in Section 2.3 from a 2D space to a Spatio-Temporal 3D space, which lays down the key for the one-pass trajectory simplification algorithms using SED (Section 4).

We consider a sub-trajectory $\ddot{\mathcal{T}}_s[P_s, \ldots, P_{s+k}]$, an error bound ϵ, and a 3D Cartesian coordinate system

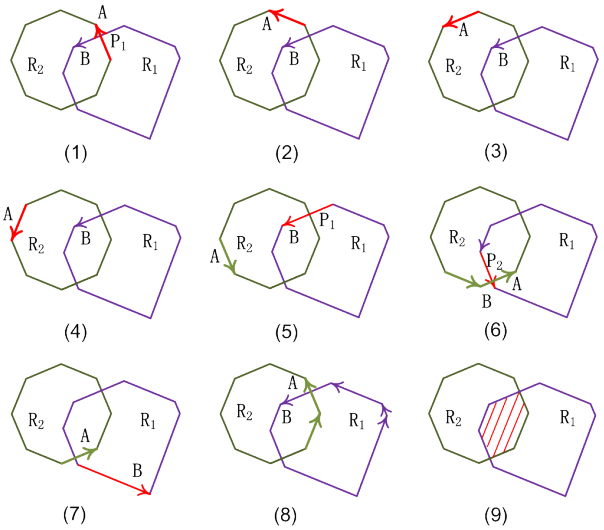

Fig. 4 A running example of convex polygons intersection.

whose origin, x-axis, y-axis and t-axis are P_s, longitude, latitude and time, respectively.

3.1 Spatio-Temporal Cone Intersection

We first present the *spatio-temporal cone intersection* method in a 3D Cartesian coordinate system, which extends the *sector intersection* method [35, 39, 40].

Synchronous Circles ($\mathcal{O}$). The synchronous circle of a data point P_{s+i} $(1 \leq i \leq k)$ in $\ddot{\mathcal{T}}_s$ *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.

Figure 5 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 know that for any point in the area of a circle $\mathcal{O}(P_{s+i}, \epsilon)$, its distance to P_{s+i} is no greater than ϵ.

Spatio-temporal cones ($\mathcal{C}$). The spatio-temporal cone (or simply *cone*) of a data point P_{s+i} $(1 \leq i \leq k)$ in $\ddot{\mathcal{T}}_s$ *w.r.t.* a point P_s and an error bound ϵ, denoted as $\mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon))$, 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.

Figure 5 also illustrates two example spatio-temporal 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.

Proposition 1: *Given a sub-trajectory* $[P_s, \ldots, P_{s+k}]$ *and a point* Q *in the area of synchronous circle* $\mathcal{O}(P_{s+k}, \epsilon)$*, the intersection point* P'_{s+i} *of the directed line segment* $\overrightarrow{P_sQ}$ *and the plane* $P.t - P_{s+i}.t = 0$ *is the synchronized point of* P_{s+i} $(1 \leq i \leq k)$ w.r.t. $\overrightarrow{P_sQ}$,

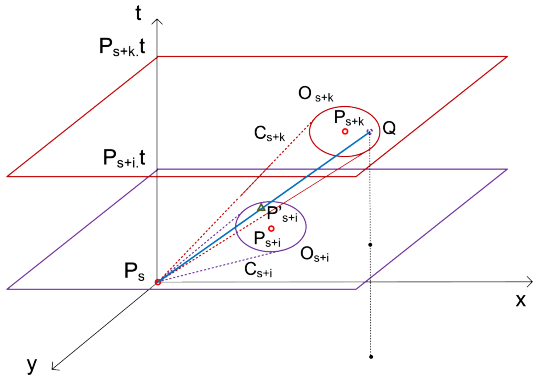

Fig. 5 Examples of spatio-temporal cones in a 3D Cartesian coordinate system, where (1) P_s, P_{s+i} and P_{s+k} are three points, (2) $\mathcal{O}_{s+i}$ and $\mathcal{O}_{s+k}$ are two synchronous circles, (3) $\mathcal{C}_{s+i}$ and $\mathcal{C}_{s+k}$ are two spatio-temporal cones, (4) Q is a point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

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* $\overrightarrow{P_sQ}$. □

Proof It suffices to show that P'_{s+i} is indeed a synchronized point P_{s+i} *w.r.t.* $\overrightarrow{P_sQ}$. The intersection point P'_{s+i} satisfies that $P'_{s+i}.t = P_{s+i}.t$ and $\frac{P'_{s+i}.t-P_s.t}{Q.t-P_s.t} = \frac{P_{s+i}.t-P_s.t}{Q.t-P_s.t} = \frac{|\overrightarrow{P_sP'_{s+i}}|}{|\overrightarrow{P_sQ}|} = \frac{P'_{s+i}.x-P_s.x}{Q.x-P_s.x} = \frac{P'_{s+i}.y-P_s.y}{Q.y-P_s.y}$. Hence, by the definition of synchronized points, we have the conclusion. □

Proposition 2: *Given a sub-trajectory* $[P_s, ..., P_{s+k}]$ *and an error bound* ϵ*, there exists a point* Q *such that* $Q.t = P_{s+k}.t$ *and* $sed(P_{s+i}, \overrightarrow{P_sQ}) \leq \epsilon$ *for each* $i \in [1, k]$ *if and only if* $\bigcap_{i=1}^{k} \mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon)) \neq \{P_s\}$. □

Proof Let P'_{s+i} ($i \in [1, k]$) be the intersection point of line segment $\overrightarrow{P_sQ}$ and the plane $P.t - P_{s+i}.t = 0$. By Proposition 1, P'_{s+i} is the synchronized point of P_{s+i} *w.r.t.* $\overrightarrow{P_sQ}$.

Assume first that $\bigcap_{i=1}^{k} \mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon)) \neq \{P_s\}$. Then there must exist a point Q in the area of the synchronous circle $\mathcal{O}(P_{s+k}, \epsilon)$ such that $\overrightarrow{P_sQ}$ 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 $sed(P_{s+i}, \overrightarrow{P_sQ}) = |\overrightarrow{P'_{s+i}P_{s+i}}| \leq \epsilon$ 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 $sed(P_{s+i}, \overrightarrow{P_sQ}) \leq \epsilon$ 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 $\bigcap_{i=1}^{k} \mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon)) \neq \{P_s\}$. □

By Proposition 2, we now have a spatio-temporal cone intersection method in a 3D Cartesian coordinate system, which significantly extends the sector intersection method [35, 39, 40] from a 2D space to a Spatio-Temporal 3D space.

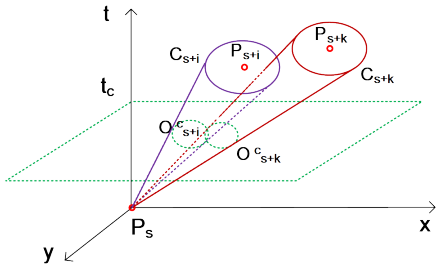

Fig. 6 Cone projection circles.

3.2 Circle Intersection

For spatio-temporal cones with the same apex P_s, the checking of their intersection can be computed by a much simpler way, *i.e.,* the checking of intersection of cone projection circles on a plane, as follows.

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) $P^c_{s+i}.x = P_s.x + c \cdot (P_{s+i}.x - P_s.x)$, (2) $P^c_{s+i}.y = P_s.y + c \cdot (P_{s+i}.y - P_s.y)$, (3) $P^c_{s+i}.t = t_c$ and (4) $r^c_{s+i} = c \cdot \epsilon$, where $c = \frac{t_c - P_s.t}{P_{s+i}.t - P_s.t}$.

In Figure 6, 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 3: *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* $sed(P_{s+i}, \overrightarrow{P_sQ}) \leq \epsilon$ *for all points* P_{s+i} *($i \in [1, k]$) if and only if* $\bigcap_{i=1}^{k} \mathcal{O}^c(P^c_{s+i}, r^c_{s+i}) \neq \emptyset$. □

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

Proposition 3 tells us that the intersection checking of spatio-temporal 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 $O(n \log n)$ [33], which cannot be used for designing one-pass trajectory simplification

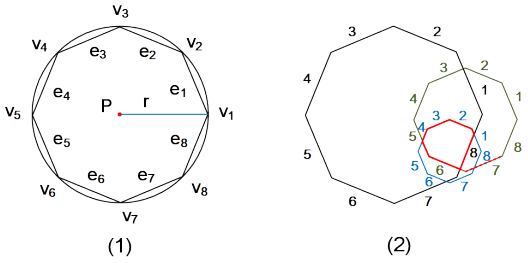

Fig. 7 Regular octagons and their intersections ($m = 8$).

algorithms using SED. However, we can approximate a circle with its m-edge inscribed regular polygon, whose intersection can be computed more efficiently.

Inscribed regular polygons ($\mathcal{R}$). Given a cone projection circle $\mathcal{O}^c(P, r)$, its inscribed m-edge 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 subscript of their start points.

Figure 7.(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 groups 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 Figure 7.(2).

Proposition 4: *The intersection $\mathcal{R}^*_l \bigcap \mathcal{R}_{s+l+1}$ $(1 \le l < k)$ has at most m edges,* i.e., *at most one from each edge group.* □

Proof We shall prove this by contradiction. Assume that $\mathcal{R}^*_l \bigcap \mathcal{R}_{s+l+1}$ has two distinct edges $\overrightarrow{A_i}$ and $\overrightarrow{A_{i'}}$ with the same label j $(1 \le j \le m)$, originally from $\mathcal{R}_{s+i}$ and $\mathcal{R}_{s+i'}$ $(1 \le i < i' \le l+1)$. Note that here $\mathcal{R}_{s+i} \bigcap \mathcal{R}_{s+i'} \neq \emptyset$ since $\mathcal{R}^*_l \bigcap \mathcal{R}_{s+l+1} \neq \emptyset$. However, when $\mathcal{R}_{s+i} \bigcap \mathcal{R}_{s+i'} \neq \emptyset$, the intersection $\mathcal{R}_{s+i} \bigcap \mathcal{R}_{s+i'}$ cannot have both edge $\overrightarrow{A_i}$ and edge $\overrightarrow{A_{i'}}$, which contradicts the assumption. □

Figure 7.(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.

Proposition 5: *The intersection of $\mathcal{R}^*_l$ and $\mathcal{R}_{s+l+1}$ $(1 \le l < k)$ can be done in $O(1)$ time.* □

Proof 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 4. As the intersection of two m-edge 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. □

3.4 Speedup Inscribed Regular Polygon Intersection

Observe that algorithm CPolyInter in Figure 3 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, which allows us to develop a fast method to compute their intersection.

Let $\vec{A} = (P_{s_A}, P_{e_A})$ and $\vec{B} = (P_{s_B}, P_{e_B})$ be two directed edges on polygons $\mathcal{R}_{s+l+1}$ and $\mathcal{R}^*_l$, respectively. Again edges $\vec{A}$ and $\vec{B}$ are moved counter-clockwise. Note that $\vec{A}$ and $\vec{B}$ are advanced step by step each time by the two advancing rules of algorithm CPolyInter. However, it is possible to advance $\vec{A}$ or $\vec{B}$ multiple steps each time. For example, in Figure 4.(1)–(5), edge $\vec{A}$ successively moves four steps, each under the advance rule (1) "$(\vec{A} \times \vec{B} < 0$ and $P_{e_A} \notin \mathcal{H}(\vec{B}))$ or $(\vec{A} \times \vec{B} \ge 0$ and $P_{e_B} \in \mathcal{H}(\vec{A}))$" of algorithm CPolyInter. Alternatively, we can directly move A from Figure 4.(1) to Figure 4.(5), by reducing four steps to one step only.

Proposition 6: *If either $(\vec{A} \bigcap \vec{B} \neq \emptyset$* **and** *$\vec{A} \times \vec{B} < 0$* **and** *$P_{e_A} \notin \mathcal{H}(\vec{B}))$ or $(\vec{A} \bigcap \vec{B} \neq \emptyset$* **and** *$\vec{A} \times \vec{B} \ge 0$* **and** *$P_{e_B} \in \mathcal{H}(\vec{A}))$ holds, then $\vec{A}$ advances s steps such that*

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

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

Proof We first explain how the edge $\vec{A}$ advances. Indeed, $\vec{A}$ is moved from its original position to its symmetric edge on $\mathcal{R}_{s+l+1}$ *w.r.t.* the symmetric line that is perpendicular to $\vec{B}$ on $\mathcal{R}^*_l$. For example, in Figure 8.(1), there is $\vec{A} \bigcap \vec{B} \neq \emptyset$ **and** $\vec{A} \times \vec{B} \ge 0$ **and** $P_{e_B} \in \mathcal{H}(\vec{A})$, hence $\vec{A}$ moves on. As $g(\vec{B}) = 3 > 1 = g(\vec{A})$, $\vec{A}$ moves forward $2 \times (g(\vec{B}) - g(\vec{A})) = 2 \times (3 - 1) = 4$ steps. Here, the label of edge $\vec{A}$ is changed to 5, its symmetric edge 1 on $\mathcal{R}_{s+l+1}$ *w.r.t.* the symmetric line that is perpendicular to $\vec{B}$ labeled with 3 on $\mathcal{R}^*_l$.

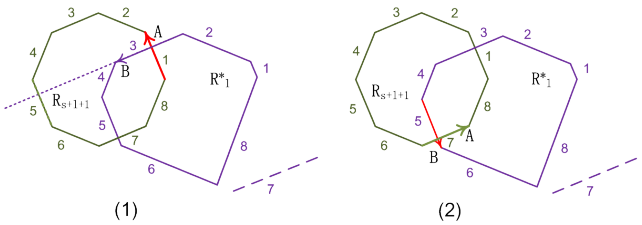

Fig. 8 Examples of fast advancing rules.

We then present the proof. If ($\overrightarrow{A} \bigcap \overrightarrow{B} \neq \emptyset$ **and** $\overrightarrow{A} \times \overrightarrow{B} < 0$ **and** $P_{e_A} \notin \mathcal{H}(\overrightarrow{B})$) or ($\overrightarrow{A} \bigcap \overrightarrow{B} \neq \emptyset$ **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 groups 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) $\overrightarrow{A} \bigcap \overrightarrow{B} = \emptyset$, 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 Section 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. □

Proposition 7: *If either* ($\overrightarrow{A} \bigcap \overrightarrow{B} \neq \emptyset$ **and** $\overrightarrow{A} \times \overrightarrow{B} \geq 0$ **and** $P_{e_B} \notin \mathcal{H}(\overrightarrow{A})$) *or* ($\overrightarrow{A} \bigcap \overrightarrow{B} \neq \emptyset$ **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* $\overrightarrow{A}$. □

Proof We first explain how the edge $\overrightarrow{B}$ is moved forward. For example, in Figure 8.(2), $\overrightarrow{A} \bigcap \overrightarrow{B} \neq \emptyset$ **and** $\overrightarrow{A} \times \overrightarrow{B} < 0$ **and** $P_{e_A} \in \mathcal{H}(\overrightarrow{B})$, hence $\overrightarrow{B}$ is moved 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 were not actually existing 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 the proof. If ($\overrightarrow{A} \bigcap \overrightarrow{B} \neq \emptyset$ **and** $\overrightarrow{A} \times \overrightarrow{B} \geq 0$ **and** $P_{e_B} \notin \mathcal{H}(\overrightarrow{A})$) or ($\overrightarrow{A} \bigcap \overrightarrow{B} \neq \emptyset$ **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) $\overrightarrow{A} \bigcap \overrightarrow{B} = \emptyset$, 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 Section 2.4, edge $\overrightarrow{B}$ is always moved forward until it reaches the target position. From this, we have the conclusion. □

Algorithm FastRPolyInter. The presented regular polygon intersection algorithm, *i.e.,* FastRPolyInter, is the optimized version of the convex polygon intersection algorithm CPolyInter, by Propositions 6 and 7. We also save vertexes of a polygon in a fixed size array, which is different from CPolyInter that saves 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 vertex by its label.

Given intersection polygon $\mathcal{R}^*_l$ of the preview l polygons and the next approximate polygon $\mathcal{R}_{s+l+1}$, the algorithm FastRPolyInter returns $\mathcal{R}^*_{l+1} = \mathcal{R}^*_l \bigcap \mathcal{R}_{s+l+1}$. 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.,* $\overrightarrow{A} \bigcap \overrightarrow{B} \neq \emptyset$ and $\overrightarrow{A} \bigcap \overrightarrow{B} = \emptyset$, where the former applies Propositions 6 and 7, and the later 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 fastens the advancing speed of directed edges $\overrightarrow{A}$ and $\overrightarrow{B}$ under certain circumstances as shown by Propositions 6 and 7, which together ensure the correctness of FastRPolyInter. Moreover, algorithm FastRPolyInter runs in $O(1)$ time by Proposition 5.

4 One-Pass Trajectory Simplification

Following [15,38], we consider two classes of trajectory simplification. The first one, referred to as *strong simplification*, that takes as input a trajectory $\ddot{\mathcal{T}}$, an error bound ϵ and the number m of edges for inscribed regular polygons, and produces a simplified trajectory $\ddot{\mathcal{T}}'$ such that all data points in $\ddot{\mathcal{T}}'$ belong to $\ddot{\mathcal{T}}$. The second one, referred to as *weak simplification*, that takes as input a trajectory $\ddot{\mathcal{T}}$, an error bound ϵ and the number m of edges for inscribed regular polygons, and produces a simplified trajectory $\ddot{\mathcal{T}}'$ such that some data points in $\ddot{\mathcal{T}}'$ may not belong to $\ddot{\mathcal{T}}$. That is, weak simplification allows data interpolation.

The main result here is stated as follows.

Theorem 8 *There exist 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 for both strong and weak trajectory simplifications, by employing the constant time synchronous distance checking technique developed in Section 3.

4.1 Strong Trajectory Simplification

Recall that in Propositions 2 and 3, the point Q may not be in the input sub-trajectory $[P_s, ..., P_{s+k}]$. If we

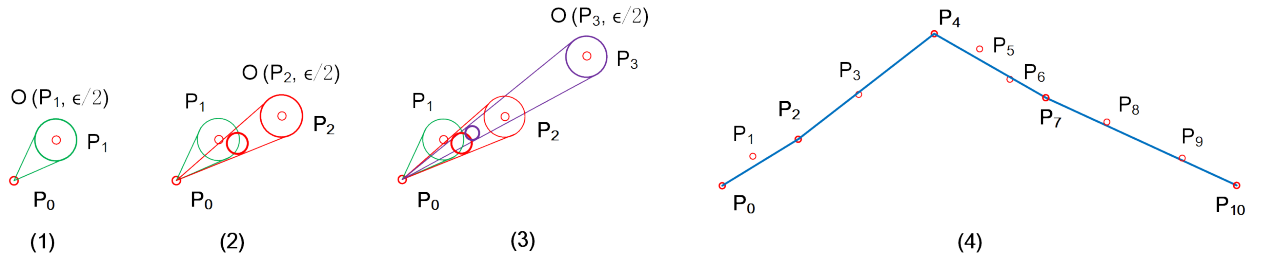

Fig. 9 A running example of the CISED-S algorithm. The points and the oblique circular cones are projected on an x-y space. The trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ is compressed into four line segments.

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.

Proposition 9: *Given a sub-trajectory* $[P_s, ..., P_{s+k}]$ *and an error bound* ϵ, $sed(P_{s+i}, \overrightarrow{P_sP_{s+k}}) \leq \epsilon$ *for each* $i \in [1, k]$ *if* $\bigcap_{i=1}^{k} \mathcal{C}(P_s, \mathcal{O}(P_{s+i}, \epsilon/2)) \neq \{P_s\}$. □

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

We first present the one-pass error bounded *strong trajectory simplification* algorithm using SED, as shown in Figure 11.

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

The parameters P_s, P_i, r and t_c together form the projection circle $\mathcal{O}^c(P_i^c, r_i^c)$ of the spatio-temporal 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, $P_i^c.x$ and $P_i^c.y$ are computed (lines 1–3), and $r_i^c = c \cdot r$. Then it builds and returns an m-edge inscribed 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 $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$, an error bound ϵ and the number m of edges for inscribed regular polygons, and returns a simplified trajectory $\overline{\mathcal{T}}$ of $\ddot{\mathcal{T}}$.

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-inscribed regular polygon

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

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

Fig. 11 One-pass strong trajectory simplification algorithm.

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 Section 3.4 (line 7). If the resulting intersection $\mathcal{R}^*$ is empty, then a new line segment $\overrightarrow{P_sP_{i-1}}$ is generated (lines 8–10). The process repeats until all points have been processed (line 11). After the final new line segment $\overrightarrow{P_sP_n}$ is generated (line 12), it returns the simplified piece-wise line representation $\overline{\mathcal{T}}$ (line 13).

Example 4 Figure 9 shows a running example of CISED-S for compressing the trajectory $\ddot{\mathcal{T}}$ in Figure 1.

(1) After initialization, the CISED-S algorithm reads point P_1 and builds a narrow *oblique circular cone* $\mathcal{C}(P_0, \mathcal{O}(P_1, \epsilon/2))$, taking P_0 as its apex and $\mathcal{O}(P_1, \epsilon/2)$ as its base (green dash). The *circular cone* is projected

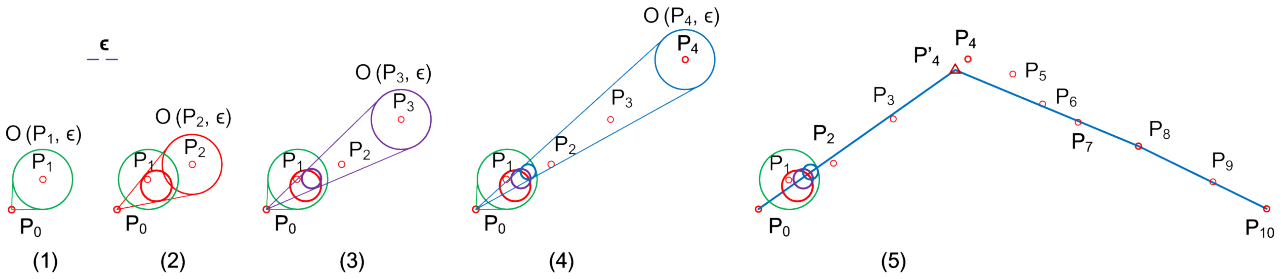

Fig. 10 A running example of the CISED-W algorithm. The points and the oblique circular cones are projected on an x-y space. The trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_{10}]$ is compressed into three line segments.

on the plane $P.t - P_1.t = 0$, and the inscribe regular polygon $\mathcal{R}_1$ of the projection circle is returned. As $\mathcal{R}^*$ is empty, $\mathcal{R}^*$ is set to $\mathcal{R}_1$.

(2) The algorithm reads P_2 and builds $\mathcal{C}(P_0, \mathcal{O}(P_2, \epsilon/2))$ (red dash). The *circular cone* is also projected on the plane $P.t - P_1.t = 0$ and the inscribe regular polygon $\mathcal{R}_2$ of the projection circle is returned. As $\mathcal{R}^* = \mathcal{R}_1$ is not empty, $\mathcal{R}^*$ is set to the intersection of $\mathcal{R}_2$ and $\mathcal{R}^*$, which is $\mathcal{R}_1 \bigcap \mathcal{R}_2 \neq \emptyset$.

(3) For point P_3, the algorithm runs the same routine as P_2 until the intersection of $\mathcal{R}_3$ and $\mathcal{R}^*$ is $\emptyset$. Thus, a line segment $\overrightarrow{P_0P_2}$ is generated, and the process of a new line segment is started, taking P_2 as the new start point and $P.t - P_3.t = 0$ as the new projection plane.

(4) At last, the algorithm outputs four continuous line segments, *i.e.*, $\{\overrightarrow{P_0P_2}, \overrightarrow{P_2P_4}, \overrightarrow{P_4P_7}, \overrightarrow{P_7P_{10}}\}$. □

4.2 Weak Trajectory Simplification

We then present the one-pass error bounded *weak simplification* algorithm using SED.

Algorithm CISED-W. Given a trajectory $\ddot{\mathcal{T}}[P_0, \ldots, P_n]$, 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 3, algorithm CISED-W generates spatio-temporal 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 $\overrightarrow{P_sP_i}$ (lines 9 and 10 of algorithm CISED-S) with Q and $\overrightarrow{P_sQ}$, respectively, such that Q is generated as follows.

Proposition 10: *Given a sub-trajectory $\ddot{\mathcal{T}}[P_s, \ldots, P_{s+k}]$ 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.* □

Proof By Proposition 3 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 $sed(P_{s+i}, \overrightarrow{P_sQ}) \leq \epsilon$ for all points P_{s+i} ($i \in [1, k]$).

From this, we have the conclusion. □

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

(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 chosen as 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 5 Figure 10 shows a running example of algorithm CISED-W for compressing the trajectory $\ddot{\mathcal{T}}$ in Figure 1 again.

(1) After initialization, the CISED-W algorithm 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 $\overrightarrow{P_0Q} = \overrightarrow{P_0P_4'}$ 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.*, $\overrightarrow{P_0P_4'}$, $\overrightarrow{P_4'P_8}$ and $\overrightarrow{P_8P_{10}}$, in which P_4' is an interpolated data points not in $\ddot{\mathcal{T}}$. □

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	~ 114.0	114M
GeoLife	182	1-5	~ 131.4	24.2M
Mopsi	51	2	~ 153.9	7.9M
PrivateCar	10	1	~ 11.8	112.8K

Correctness and complexity analyses. The correctness of algorithms CISED-S and CISED-W follows from Propositions 3 and 9, and Propositions 3 and 10, 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 optimal algorithm using SED and existing algorithms of DPSED and SQUISH-E on trajectory datasets. Using four real-life trajectory datasets, we conducted three sets of experiments to evaluate: (1) the compression ratios of algorithms CISED-S and CISED-W vs. DPSED, SQUISH-E and the optimal algorithm, (2) the average errors of algorithms CISED-S and CISED-W vs. DPSED, SQUISH-E and the optimal algorithm, (3) the execution time of algorithms CISED-S and CISED-W vs. DPSED, SQUISH-E and the optimal algorithm, and (4) the impacts of polygon intersection algorithms FastRPolyInter and CPolyInter and the edge number m of inscribed regular polygons to the effectiveness and efficiency of algorithms CISED-S and CISED-W.

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 Apr. 2015 to Nov. 2015. The sampling rate was one point per 3–5 seconds, and each trajectory has around $114.1K$ points.

(2) GeoLife trajectory data (GeoLife) is the GPS trajectories collected in GeoLife project [41] 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 in each 1-5 seconds per point.

(3) Mopsi trajectory data (Mopsi) is the GPS trajectories collected in Mopsi project [1] by 51 users in a period from 2008 to 2014. Most routes are in Joensuu region, Finland. The sampling rate was one point per 2 seconds, 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.

(5) Small trajectory data. As the optimal LS algorithm [14] it 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 (too slow and out of memory). Hence, we further build four *small datasets*, each dataset includes 10 middle-size ($10K$ points per trajectory) trajectories selected from ServiceCar, GeoLife, Mopsi and PrivateCar, respectively.

Algorithms and implementation. We implement five LS algorithms, *i.e.,* our CISED-S and CISED-W, DPSED [20] (the most effective existing LS algorithm using SED), SQUISH-E [23] (the most efficient existing LS algorithm using SED) and the optimal LS algorithm using SED (see Section 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 the optimal algorithm. The compression ratio is defined as follows: Given a set of trajectories

$\{\dddot{\mathcal{T}}_1, \ldots, \dddot{\mathcal{T}}_M\}$ and their piecewise line representations $\{\overline{\mathcal{T}_1}, \ldots, \overline{\mathcal{T}_M}\}$, the compression ratio of an algorithm is $(\sum_{j=1}^{M} |\overline{\mathcal{T}}_j|)/(\sum_{j=1}^{M} |\dddot{\mathcal{T}}_j|)$. 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, we fixed the error bounds $\epsilon = 60$ meters, and varied m from 4 to 40. The results are reported in Figure 12.

(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.

(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 the 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 meters to 200 meters on the entire four datasets, respectively. The results are reported in Figure 13 .

(1) When increasing ϵ, the compression ratios of all these algorithms decrease on all datasets.

(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-Eand comparable with DPSED on all datasets and for all ϵ. The compression ratios of CISED-S are on average (79.3%, 71.9%, 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 $\epsilon = 40$ meters, 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.

(4) Algorithm CISED-W has better compression ratios than DP, 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 $\epsilon = 40$ meters, the compression ratios of algorithm CISED-W are (11.5%, 4.3%, 2.8%, 2.7%) on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively.

Exp-1.3: Impacts of the error bound ϵ on compression ratios (VS. the optimal algorithm). To evaluate the impacts of error bound ϵ on compression ratios, we once again fixed m=16, the middle of $[12, 20]$, and varied ϵ from 10 to 200 meters on the first $1K$ points of each trajectory of the selected *small datasets*, respectively. The results are reported in Figure 14 .

(1) Algorithm CISED-S is poorer than the optimal algorithm 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 the optimal algorithm on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. For example, when $\epsilon = 40$ meters, the compression ratios of CISED-S and the optimal algorithm 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 is comparable with the optimal algorithm 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 the optimal algorithm on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. For example, when $\epsilon = 40$ meters, the compression ratios of algorithm CISED-W are (14.6%, 4.6%, 1.2%, 2.5%) on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively.

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 meters, and varied the size $|\dddot{\mathcal{T}}|$ of trajectories from $1K$ points to $10K$ points. The results are reported in Figure 15.

(1) The compression ratios of these algorithms from the best to the worst are CISED-W, DPSED, CISED-S and SQUISH-E, on all datasets and for all sizes of trajectories.

(2) The size of input trajectories has 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 av-

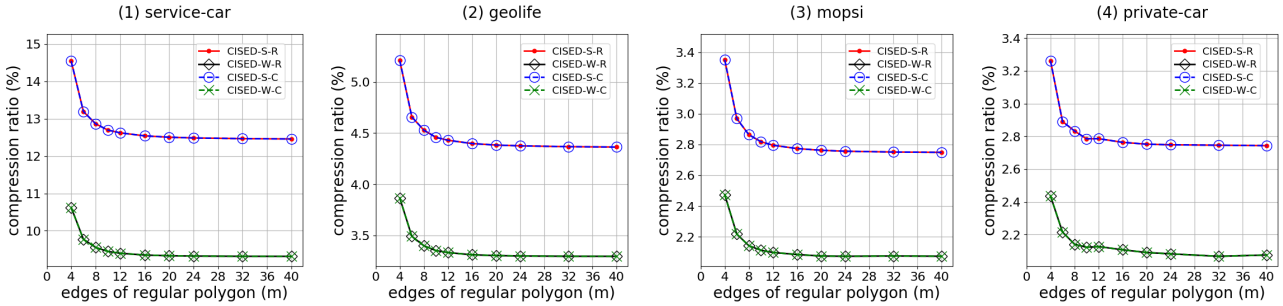

Fig. 12 Evaluation of compression ratios: fixed error bound with $\epsilon = 60$ meters and varying m.

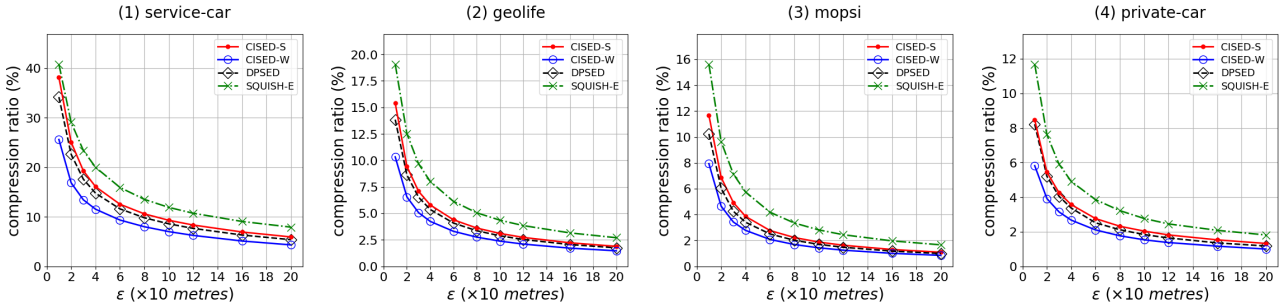

Fig. 13 Evaluation of compression ratios: fixed with $m = 16$ and varying error bound ϵ.

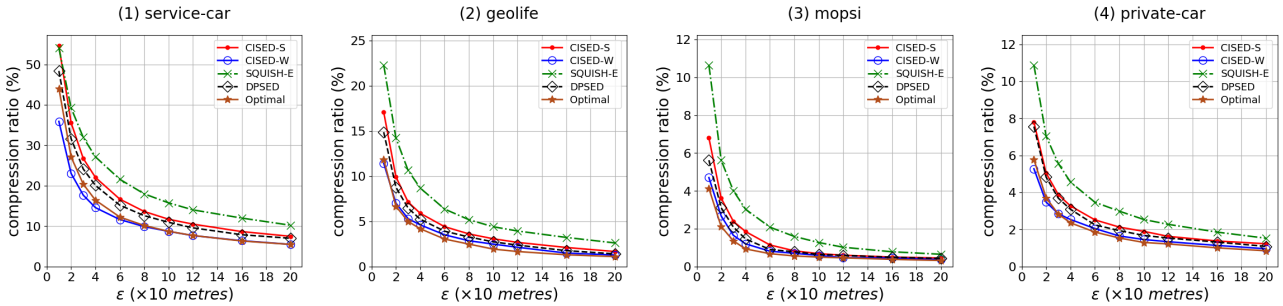

Fig. 14 Evaluation of compression ratios: fixed with $m = 16$ and varying error bound ϵ (on small datasets).

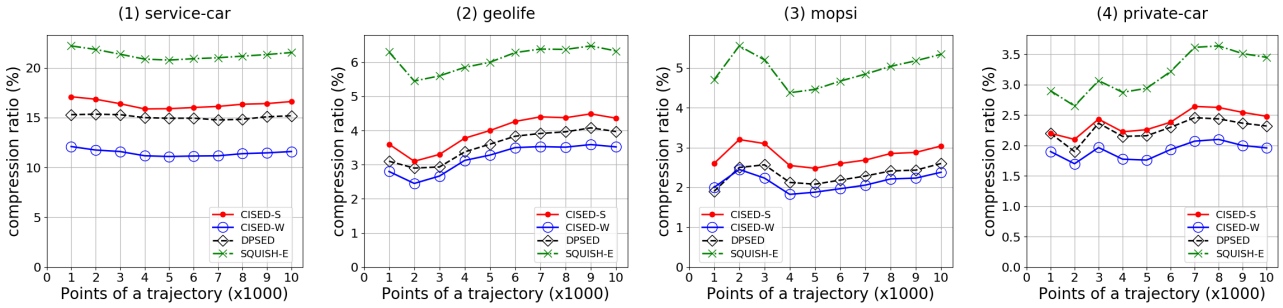

Fig. 15 Evaluation of compression ratios: fixed with $m = 16$ and $\epsilon = 60$ meters, and varying the size of trajectories.

erage errors of our algorithms CISED-S and CISED-W with DPSED, SQUISH-E and the optimal algorithm.

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, we fixed the error bounds $\epsilon = 60$ meters, and varied m from 4 to 40. The results are reported in Figure 16.

(1) Algorithms CISED-S and CISED-W using FastRPolyInter have the same average errors as

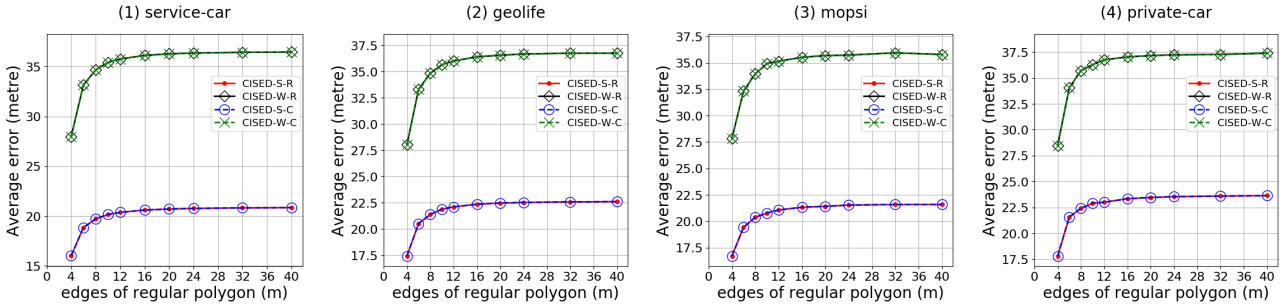

Fig. 16 Evaluation of average errors: fixed error bound with $\epsilon = 60$ meters and varying m.

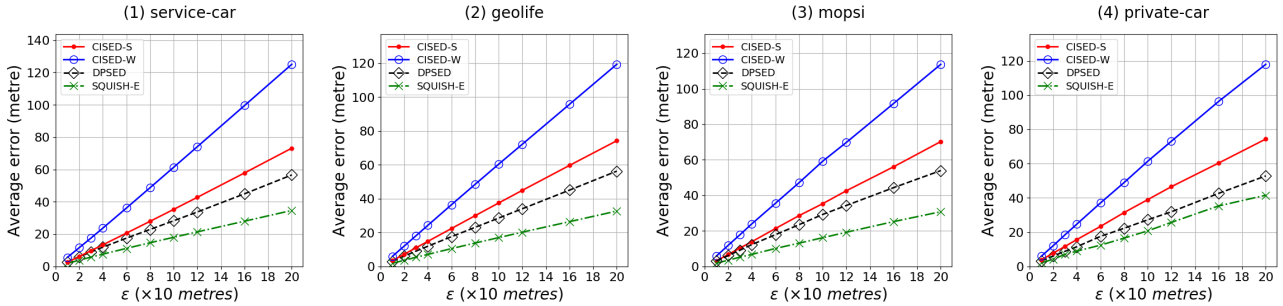

Fig. 17 Evaluation of average errors: fixed with $m = 16$ and varying error bound ϵ.

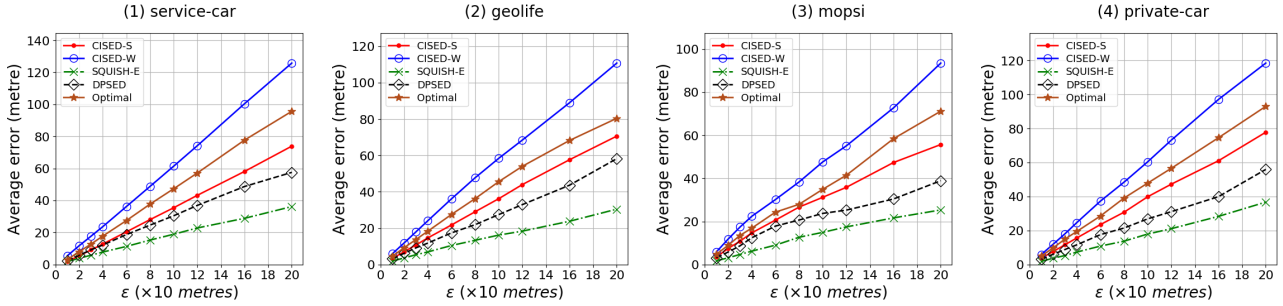

Fig. 18 Evaluation of average errors: fixed with $m = 16$ and varying error bound ϵ (on small datasets).

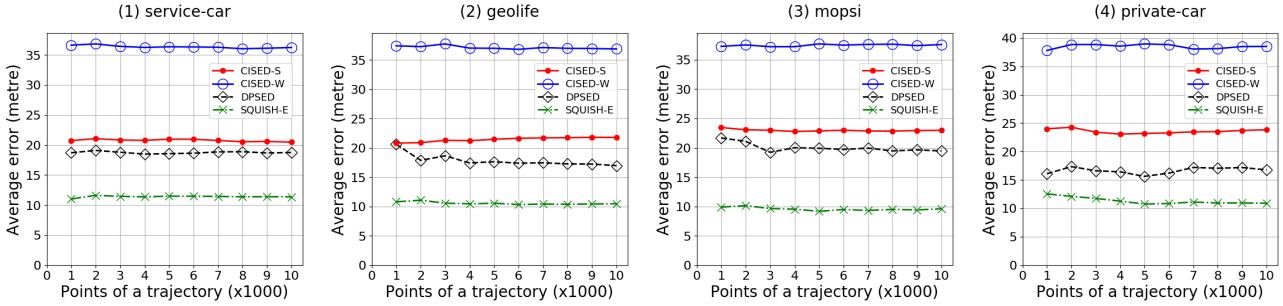

Fig. 19 Evaluation of average errors: fixed with $m = 16$ and $\epsilon = 60$ meters, and varying the size of trajectories.

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.

(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 meters to 200 meters on the entire datasets ServiceCar, GeoLife, Mopsi and PrivateCar, respectively. The results are reported in Figure 17.

(1) Average errors increase with the increase of ϵ.

(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.

(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 optimal algorithm). To evaluate the average errors of these algorithms, we once again fixed m=16, and varied ϵ from 10 meters to 200 meters on the first $1K$ points of each trajectory of the selected *small datasets*, respectively. The results are reported in Figure 18.

The average errors of these algorithms from the largest to the smallest are CISED-W, the optimal algorithm 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 the optimal algorithm on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively.

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 $\epsilon = 60$ meters, and varied the size $|\dddot{\mathcal{T}}|$ of trajectories from $1K$ points to $10K$ points. The results are reported in Figure 19.

(1) The average errors of these algorithms from the smallest to the largest are SQUISH-E, DPSED, CISED-S and CISED-W, on all datasets and for all trajectory sizes.

(2) The size of input trajectories has few impacts on the average errors of LS algorithms on all datasets.

5.2.3 Evaluation of Efficiency

In the last set of tests, we evaluate the impacts of parameter m on the efficiency of algorithms CISED-S and CISED-W, and compare the efficiency of our approaches CISED-S and CISED-W with the optimal algorithm and algorithms DPSED and SQUISH-E.

Exp-3.1: Impacts of algorithm FastRPolyInter and parameter m on efficiency. To evaluate the impacts of FastRPolyInter and parameter m on algorithm CISED-S and CISED-W, we equipped CISED-S and CISED-W with FastRPolyInter and CPolyInter, respectively, fixed $\epsilon = 60$ meters, and varied m from 4 to 40. The results are reported in Figures 20 and 21.
(1) The algorithms CISED-S and CISED-W spend the most time in the executing of 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. 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.
(4) The running time of $m = 12$ is on average 69.92% of $m = 20$ for CISED-S and CISED-W on all datasets.

Exp-3.2: Impacts of the error bound ϵ on efficiency (VS. algorithms DPSED and SQUISH-E). To evaluate the impacts of ϵ on efficiency, we fixed m = 16, and varied ϵ from 10 meters to 200 meters on the entire datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. The results are reported in Figure 22.
(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.

Exp-3.3: Impacts of the error bound ϵ on efficiency (VS. the optimal algorithm). To evaluate the impacts of ϵ on efficiency, we once again fixed m

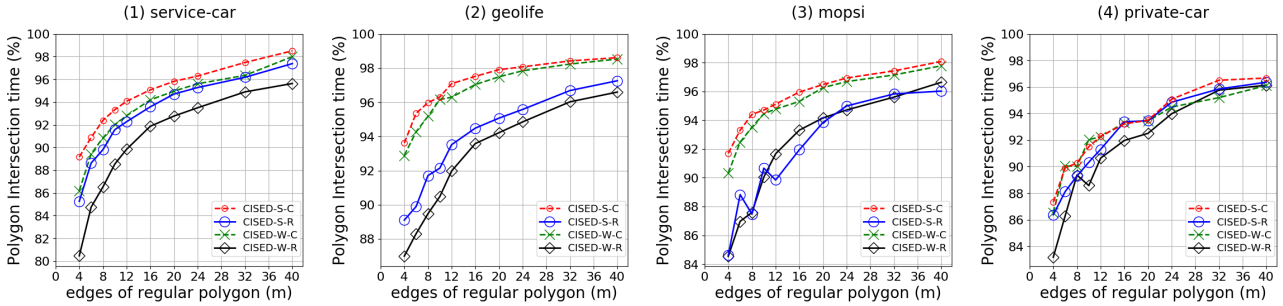

Fig. 20 Evaluation of running time of polygon intersection algorithms: fixed error bound with $\epsilon = 60$ meters, and varying m. Here "R" denotes our fast regular polygon intersection algorithm FastRPolyInter, and "C" denotes CPolyInter, respectively.

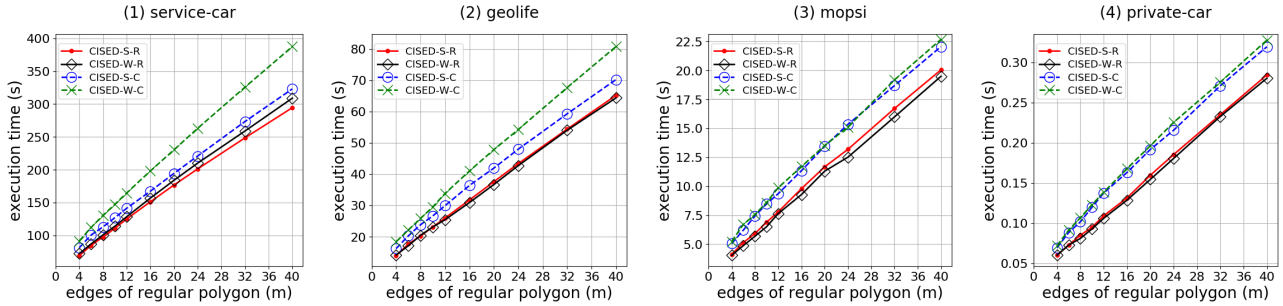

Fig. 21 Evaluation of running time: fixed error bound with $\epsilon = 60$ meters, and varying m.

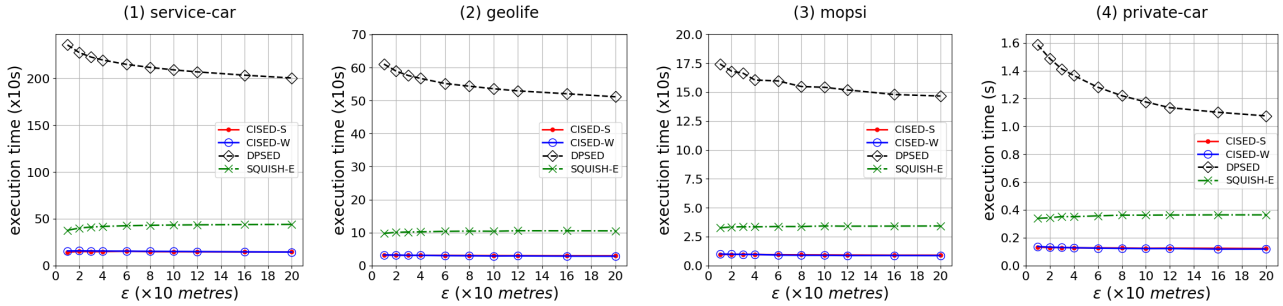

Fig. 22 Evaluation of running time: fixed with $m = 16$ and varying error bounds ϵ.

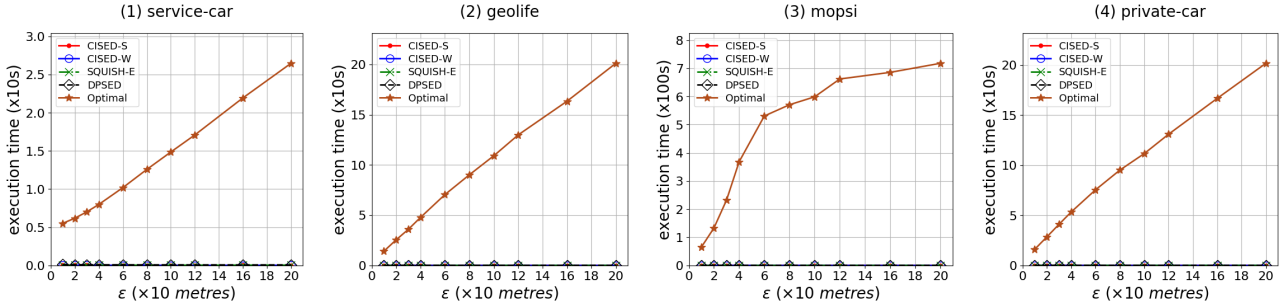

Fig. 23 Evaluation of running time: fixed with $m = 16$ and varying error bounds ϵ (on small datasets).

$= 16$, and varied ϵ from 10 meters to 200 meters on the first $1K$ points of each trajectory of the selected *small datasets*, respectively. The results are reported in Figure 23.

(1) Algorithms CISED-S and CISED-W are obviously faster than the optimal algorithm for all cases. They are on average (925.25, 7888.26, 40041.59, 8528.76) times faster than the optimal algorithm on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively.

Exp-3.4: Impacts of trajectory sizes on efficiency. To evaluate the impacts of trajectory sizes on execution time, we chose the same 10 trajectories,

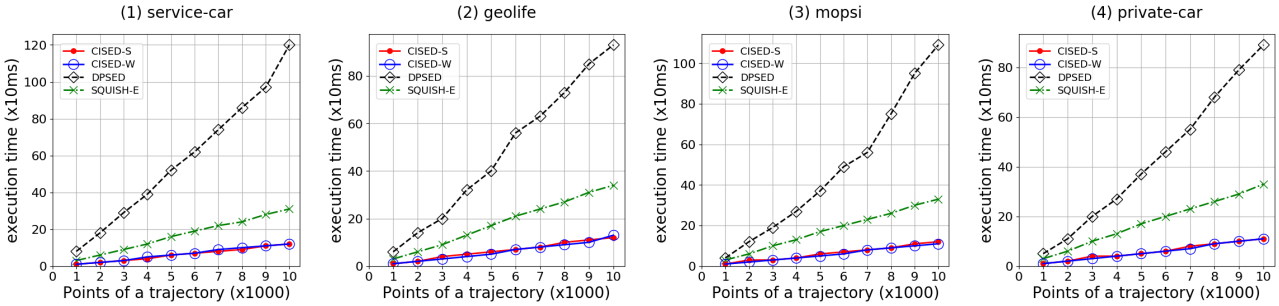

Fig. 24 Evaluation of running time: fixed with $m = 16$ and $\epsilon = 60$ meters, and varying the size of trajectories.

from datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively, fixed $m = 16$ and $\epsilon = 60$ meters, and varied the size $|\dddot{\mathcal{T}}|$ of trajectories from $1K$ points to $10K$ points. The results are reported in Figure 24.

(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 efficiency advantage of algorithms CISED-S and CISED-W increases with the increase of trajectory sizes compared with DPSED and SQUISH-E.

5.2.4 Summary

From these tests we find the following.

(1) Polygon intersection Algorithms. Algorithm FastRPolyInter runs faster than CPolyInter, and is as effective as CPolyInter.

(2) 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.

(3) Compression ratios. The optimal LS algorithm has the best compression ratios among all strong simplification algorithms. Algorithm CISED-S is comparable with DPSED and algorithm CISED-W is comparable with the optimal LS algorithm. They are all better than SQUISH-E. The compression ratios of algorithm CISED-S, the optimal algorithm and algorithm 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.

(4) Average errors. The average errors of these algorithms from the smallest to the largest are SQUISH-E, DPSED, CISED-S, the optimal LS algorithm and CISED-W. Algorithm CISED-W has obvious higher average errors than CISED-S as the former essentially forms spatio-temporal cones with a radius of ϵ.

(5) 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 the optimal LS algorithm on datasets (ServiceCar, GeoLife, Mopsi, PrivateCar), respectively. The efficiency advantage of algorithms CISED-S and CISED-W also increases with the increase of the trajectory size.

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 derivations, 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 piece-wise line simplification comes from computational geometry. Its target is to approximate a given finer piece-wise linear curve by another coarser piece-wise 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 [20], and formally presented in [28] 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 later reserves temporal information in the result trajectories.

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

6.1.1 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 [14] 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 [37] and Melkman and O'Rourke of [19] improved the time complexity to $O(n^2 \log n)$ by using either *convex hull* or *sector intersection* methods. The authors of [3] further proved that the 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 can not work with SED, hence, currently the time complexity of the optimal algorithm using SED remains $O(n^3)$. It is time-consuming and impractical for large trajectory data [11].

6.1.2 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 the aspect of the 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 [29] and Douglas-Peucker [7], 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 batch algorithms originally only support PED, but are easy to be extended to support SED [20]. The batch nature and high time complexities make batch algorithms impractical for online and resource-constrained scenarios [15].

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 [20]. These methods still have a high time and/or space complexity, which significantly hinders their utility in resource-constrained mobile devices [16]. BQS [16,17] and SQUISH-E [23] further optimize the opening window algorithms. BQS [16, 17] fasts 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 an 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 [34] 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 [30], 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 [35,39] 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. [8] optimized algorithm SI by considering the distance between a potential end point and the initial point of a line segment, and the Sleeve algorithm [40] in the cartographic discipline essentially applies the same idea as the SI algorithm. Moreover, fast BQS [16] (FBQS in short), the simplified version of BQS, has a linear time complexity. The authors of this article also developed an One-Pass ERror Bounded (OPERB) algorithm [15]. However, all existing one-pass algorithms use PED [8, 15, 16, 35, 39, 40], 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 [5,6,9,10,13,27,36] project trajectory points onto roads (also known as Map-Matching). Moreover, [9, 10, 36] mine and use high frequency patterns of compressed trajectories, instead of roads, to further improve compression effectiveness. Some methods [31, 32] 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 [31]. There are also compression algorithms preserving the direction of the trajectory [18].

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 both efficient and effective. They are three times faster than SQUISH-E, the most efficient existing LS algorithm using SED. In terms of compression ratio, algorithm CISED-S is comparable with 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 comparable with the optimal algorithm and is on average 19.6% and 42.4% better than DPSED and SQUISH-E, respectively.

Acknowledgments

This work is supported in part by NSFC (U1636210), 973 program (2014CB340300), NSFC (61421003) and Beijing Advanced Innovation Center for Big Data and Brain Computing.

References

1. Mopsi routes 2014. `http://cs.uef.fi/mopsi/routes/dataset/`. Accessed: 2017-11-29.
2. H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal data reduction with deterministic error bounds. *VLDBJ*, 15(3):211–228, 2006.
3. W. Chan and F. Chin. Approximation of polygonal curves with minimum number of line segments or minimal error. *International Journal of Computational Geometry Applicaitons*, 6(1):378–387, 1996.
4. M. Chen, M. Xu, and P. Fränti. A fast multiresolution polygonal approximation algorithm for GPS trajectory simplification. *IEEE Trans. Image Processing*, 21(5):2770–2785, 2012.
5. Y. Chen, K. Jiang, Y. Zheng, C. Li, and N. Yu. Trajectory simplification method for location-based social networking services. In *LBSN*, pages 33–40, 2009.
6. A. Civilis, C. S. Jensen, and S. Pakalnis. Techniques for efficient road-network-based tracking of moving objects. *TKDE*, 17(5):698–712, 2005.
7. D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *The Canadian Cartographer*, 10(2):112–122, 1973.
8. J. G. Dunham. Piecewise linear approximation of plannar curves. *PAMI*, 8, 1986.
9. R. Gotsman and Y. Kanza. A dilution-matching-encoding compaction of trajectories over road networks. *GeoInformatica*, 2015.
10. Y. Han, W. Sun, and B. Zheng. Compress: A comprehensive framework of trajectory compression in road networks. *TODS*, 42(2):11:1–11:49, 2017.
11. P. S. Heckbert and M. Garland. Survey of polygonal surface simplification algorithms. In *SIGGRAPH*, 1997.
12. J. Hershberger and J. Snoeyink. Speeding up the douglas-peucker line-simplification algorithm. *Technical Report, University of British Columbia*, 1992.
13. C. C. Hung, W. Peng, and W. Lee. Clustering and aggregating clues of trajectories for mining trajectory patterns and routes. *VLDBJ*, 24(2):169–192, 2015.
14. H. Imai and M. Iri. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision Graphics and Image Processing*, 36:31–41, 1986.
15. X. Lin, S. Ma, H. Zhang, T. Wo, and J. Huai. One-pass error bounded trajectory simplification. *PVLDB*, 10(7):841–852, 2017.
16. J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, and R. Jurdak. Bounded quadrant system: Error-bounded trajectory compression on the go. In *ICDE*, 2015.
17. J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, J.-G. Lee, and R. Jurdak. A novel framework for online amnesic trajectory compression in resource-constrained environments. *IEEE Transactions on Knowledge and Data Engineering*, 28(11):2827–2841, 2016.
18. C. Long, R. C.-W. Wong, and H. Jagadish. Direction-preserving trajectory simplification. *PVLDB*, 6(10):949–960, 2013.

19. A. Melkman and J. O'Rourke. On polygonal chain approximation. *Machine Intelligence and Pattern Recognition*, 6:87–95, 1988.
20. N. Meratnia and R. A. de By. Spatiotemporal compression techniques for moving point objects. In *EDBT*, 2004.
21. R. Metha and V.K.Mehta. *The Principles of Physics*. S Chand, 1999.
22. J. Muckell, J.-H. Hwang, C. T. Lawson, and S. S. Ravi. Algorithms for compressing gps trajectory data: an empirical evaluation. In *ACM-GIS*, 2010.
23. J. Muckell, P. W. Olsen, J.-H. Hwang, C. T. Lawson, and S. S. Ravi. Compression of trajectory data: a comprehensive evaluation and new approach. *GeoInformatica*, 18(3):435–460, 2014.
24. A. Nibali and Z. He. Trajic: An effective compression system for trajectory data. *TKDE*, 27(11):3138–3151, 2015.
25. J. O'Rourke, C. B. Chien, T. Olson, and D. Naddor. A new linear algorithm for intersecting convex polygons. *Computer Graphics and Image Processing*, 19(4):384–391, 1982.
26. T. Pavlidis and S. L. Horowitz. Segmentation of plane curves. *IEEE Transactions on Computers.*, 23(8):860–870, 1974.
27. I. S. Popa, K. Zeitouni, VincentOria, and A. Kharrat. Spatio-temporal compression of trajectories in road networks. *GeoInformatica*, 19(1):117–145, 2014.
28. M. Potamias, K. Patroumpas, and T. K. Sellis. Sampling trajectory streams with spatiotemporal criteria. In *SSDBM*, 2006.
29. U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Comput. Graphics Image Processing*, 1:244–256, 1972.
30. K. Reumann and A. Witkam. Optimizing curve segmentation in computer graphics. In *International Computing Symposium*, 1974.
31. K.-F. Richter, F. Schmid, and P. Laube. Semantic trajectory compression: Representing urban movement in a nutshell. *Journal of Spatial Information Science*, 4(1):3–30, 2012.
32. F. Schmid, K. Richter, and P. Laube. Semantic trajectory compression. In *SSTD*, pages 411–416, 2009.
33. Shamos, M. Ian, and H. Dan. Geometric intersection problems. In *Symposium on Foundations of Computer Science*, pages 208–215, 1976.
34. W. Shi and C. Cheung. Performance evaluation of line simplification algorithms for vector generalization. *Cartographic Journal*, 43(1):27–44, 2006.
35. J. Sklansky and V. Gonzalez. Fast polygonal approximation of digitized curves. *Pattern Recognition*, 12:327–331, 1980.
36. R. Song, W. Sun, B. Zheng, and Y. Zheng. Press: A novel framework of trajectory compression in road networks. *PVLDB*, 7(9):661–672, 2014.
37. G. T. Toussaint. On the complexity of approximating polygonal curves in the plane. In *International Symposium on Robotics and Automation (IASTED)*, 1985.
38. G. Trajcevski, H. Cao, P. Scheuermanny, O. Wolfsonz, and D. Vaccaro. On-line data reduction and the quality of history in moving objects databases. In *MobiDE*, 2006.
39. C. M. Williams. An efficient algorithm for the piecewise linear approximation of planar curvers. *Computer Graphics and Image Processing*, 8:286–293, 1978.
40. Z. Zhao and A. Saalfeld. Linear-time sleeve-fitting polyline simplification algorithms. In *Proceedings of AutoCarto*, pages 214–223, 1997.
41. Y. Zheng, X. Xie, and W. Ma. GeoLife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, 33(2):32–39, 2010.