

LiMITS: An Effective Approach for Trajectory Simplification

Yunheng Han
The University of Maryland
College Park, Maryland
yhhan@cs.umd.edu

Hanan Samet
The University of Maryland
College Park, Maryland
hjs@cs.umd.edu

ABSTRACT

Trajectories represent the mobility of moving objects and thus is of great value in data mining applications. However, trajectory data is enormous in volume, so it is expensive to store and process the raw data directly. Trajectories are also redundant so data compression techniques can be applied. In this paper, we propose effective algorithms to simplify trajectories. We first extend existing algorithms by replacing the commonly used L_2 metric with the L_∞ metric so that they can be generalized to high dimensional space (e.g., 3-space in practice). Next, we propose a novel approach, namely **L-infinity Multidimensional Interpolation Trajectory Simplification (LiMITS)**. LiMITS belongs to weak simplification and takes advantage of the L_∞ metric. It generates simplified trajectories by multidimensional interpolation. It also allows a new format called compact representation to further improve the compression ratio. Finally, We demonstrate the performance of LiMITS through experiments on real-world datasets, which show that it is more effective than other existing methods.

ACM Reference Format:

Yunheng Han and Hanan Samet. 2018. LiMITS: An Effective Approach for Trajectory Simplification. In *Woodstock '18: ACM Symposium on Neural Gaze Detection, June 03–05, 2018, Woodstock, NY*. ACM, New York, NY, USA, 13 pages. <https://doi.org/10.1145/1122445.1122456>

1 INTRODUCTION

Trajectory data is ubiquitous and informative. Nowadays GPS navigation devices are constantly generating unprecedented amounts of trajectory data, from cell phones in hands to vehicles on the way. The enormous amounts of data make it possible to discover knowledge effectively as trajectories represent the mobility of the moving objects, e.g., many data driven approaches have been proposed to solve real-world problems in the past decades [46]. However, such huge amounts also result in difficulty in data storage and retrieval, which influences the performance of all algorithms based on real trajectories. Trajectory data is also redundant because of inertia. Given the historical trajectory of a moving object, it is most

This work was supported in part by the US National Science Foundation under grants IIS-13-20791 and IIS-1816889.

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

Woodstock '18, June 03–05, 2018, Woodstock, NY

© 2018 Association for Computing Machinery.

ACM ISBN 978-1-4503-XXXX-X/18/06...\$15.00

<https://doi.org/10.1145/1122445.1122456>

likely that it keeps the current state (e.g., displacement, velocity and acceleration). The probability distribution of the trajectory, therefore, has relatively low information entropy, and data compression techniques can be adopted to reduce the storage cost.

Data compression could be either *lossless* or *lossy*. Lossless compression is widely used for text and images [44, 48], but it has been proved to be ineffective for trajectory data [9, 11, 29, 45]. In practice, most applications do not require as much accuracy as raw data has. As a result, many lossy approaches were proposed, namely trajectory simplification [10, 15, 21, 26, 27]. These approaches represent trajectories with fewer points while a certain data quality is still preserved. The output of trajectory simplification shares the same format as the input, so it is as convenient to process the simplified data. Furthermore, information retrieval is even more efficient on the simplified data: the higher the compression ratio is, the fewer points the simplified data contains, and thus the less time query processing costs. The simplification algorithms are classified into two categories, *strong* and *weak simplification*. Strong simplification produces a subset of the input data, while weak simplification may involve new points. Weak simplification is more challenging because the number of feasible solutions becomes uncountable (while it is finite in strong simplification). On the other hand, it could be more effective and hence a direction worth exploring.

The quality of trajectory data after simplification is guaranteed by the similarity between input and output trajectories: the more similar input and output are, the higher the quality [41]. A trajectory is the path of a moving object in space as a function of time, and thus it contains both spatial and temporal information. In order to measure the similarity between trajectories, Fréchet distance [1, 5] is usually used because it takes into account not only the spatial information—the locations of the points, but also the temporal information—the order they follow. In the scope of trajectory simplification, the error is often bounded by the *synchronized distance* [7, 26], a specialized version of the general Fréchet distance, which is defined as

$$d(S, T) = \max_t \|S(t) - T(t)\| \quad (1)$$

where S, T are two trajectories, t is a time stamp and $\|x - y\|$ is the distance between x and y . It measures distance between two objects at all time stamps and takes the maximum as the result. If $d(S, T)$ is bounded by a given threshold, i.e., $d(S, T) \leq \epsilon$, the derivation of output from input at any time stamp will be no greater than ϵ , so the data quality is preserved. It is crucial to design a suitable and effective distance metric, as the problem of trajectory simplification can be perceived an instance of similarity learning [45]. Existing work considers the *Euclidean distance* (L_2) in Equation 1, since it is the ordinary straight line distance. Notwithstanding, there are many other distance metrics, among which the Minkowski distance is an extension of the L_2 distance. For any value of p , the Minkowski

distance is defined as

$$\|\mathbf{x} - \mathbf{y}\| = \left(\sum_{i=1}^m |x_i - y_i|^p \right)^{\frac{1}{p}}. \quad (2)$$

Note that only when $p \geq 1$ does the triangle inequality hold. Moreover, it is known as the *Manhattan distance* or the city block distance (L_1) when $p = 1$, and the *Chebyshev distance* or the chessboard distance (L_∞) when $p \rightarrow \infty$. Different from existing work, we use the *synchronized L_∞ distance* to bound the similarity between two trajectories.

L_∞ is widely used in many applications including quadtree image representations [30, 31], corridor computation [2, 3], polygonal curve approximation [12, 14], warehouse systems [18] and computer aided manufacturing [35]. However, it has not been explored in the problem of trajectory simplification. L_∞ is advantageous for the following reasons. First, it has the same properties as other Minkowski distances, such as the triangle inequality and translation invariance. Second, L_∞ is a good approximation to other metrics. Third, it is the limiting case in Equation 2 and thus smaller than any other distances, so it leads to the loosest error bounds and the highest compression ratio. Finally, L_∞ makes it possible to extend existing simplification algorithms [11, 15, 21, 26, 38] originally designed for trajectories in 1-space and 2-space, including the optimal strong simplification and state-of-the-art weak simplification, to high dimensions (e.g., 3-space in practice), while it is difficult under other metrics. Hence L_∞ is suitable for real applications on trajectory data.

In this paper, we propose an effective approach for trajectory simplification, namely *L-infinity Multidimensional Interpolation Trajectory Simplification* (LiMITS). It is weak simplification algorithm under the L_∞ metric. Here we only present an overview of LiMITS. Given a trajectory T in m -space, LiMITS first projects it onto the planes spanned by the time axis and every spatial axis, forming m trajectories T_1, T_2, \dots, T_m in 1-space. Since trajectory simplification in 1-space is feasible, we simplify them individually and obtain simplified trajectories S_i for each T_i . Eventually, we combine S_1, S_2, \dots, S_m by interpolation to construct an output trajectory S in m -space. Note that the straightforward implementation is simple but not effective, and we leave the elaboration to the later sections. To sum up, the main contributions of our work include:

- (1) We explore the L_∞ distance for trajectory simplification and extend existing algorithms to higher dimensions. This is crucial to real applications as most trajectories are in 3-space.
- (2) LiMITS falls into the category of weak simplification, in which the constraints on the output with respect to the input are fewer, and thus is more effective than existing methods in terms of compression ratio.
- (3) Besides the normal representation where trajectories are sequences of points, LiMITS allows a compact representation leading to an even higher compression ratio.
- (4) LiMITS has a linear time complexity and hence is efficient for arbitrarily long trajectories.
- (5) We conducted extensive experiments on real-world trajectory datasets generated by humans and animals in 2-space and 3-space. The results show that the compression ratio of our approach is higher than that of existing algorithms.

The rest of the paper is organized as follows. Section 2 introduces preliminaries and formalizes the problem of trajectory simplification. Section 3 proposes our approach based on multidimensional interpolation. Section 4 reports our experiments with real data to demonstrate the performance of the algorithms. Section 5 reviews related work and compares existing techniques with our approach. Section 6 summarizes the paper and discusses directions for future research.

2 PRELIMINARIES

In this section, we present the formal definition of the problem. We first start with some preliminaries. According to the definition, a trajectory is the path of a moving object in space as a function of time. The path contains an infinite number of locations, but only a few of the locations are sampled in practice. Hence trajectory data is usually represented by a series of sample points:

$$T = \langle \mathbf{p}_1, t_1 \rangle, \langle \mathbf{p}_2, t_2 \rangle, \dots, \langle \mathbf{p}_n, t_n \rangle,$$

where T is the trajectory and n is the length. Moreover, the time stamps should increase strictly, i.e., $t_i < t_{i+1}$. This representation does not show the information of the locations between two consecutive time stamps since they are not sampled. Nonetheless, we are able to recover them through linear interpolation and the interpolation tends to be accurate when the sampling rate increases. A point of the trajectory in m -space is given by m coordinate values:

$$T(t) = \langle T_1(t), T_2(t), \dots, T_m(t) \rangle.$$

The k -th coordinate value of the interpolation point at any time t is calculated as

$$T_k(t) = T_k(t_i) + \frac{T_k(t_{i+1}) - T_k(t_i)}{t_{i+1} - t_i} \cdot (t - t_i),$$

where $1 \leq k \leq m$, $1 \leq i < n$ and $t_i \leq t \leq t_{i+1}$ (see Figure 1, where $i = 4$). This can be done in all dimensions and the interpolation point in m -space is

$$T(t) = T(t_i) + \frac{T(t_{i+1}) - T(t_i)}{t_{i+1} - t_i} \cdot (t - t_i), \quad (3)$$

where $T(t_i) = \mathbf{p}_i$ and $T(t_{i+1}) = \mathbf{p}_{i+1}$ are sample points. For example, a location in 2-space can be represented by (x_i, y_i) , so a trajectory will be in the form of $\langle x_1, y_1, t_1 \rangle, \langle x_2, y_2, t_2 \rangle, \dots, \langle x_n, y_n, t_n \rangle$. Furthermore, the location $T(t) = \langle x(t), y(t) \rangle$ at an arbitrary time stamp t is given by

$$x(t) = x_i + \frac{x_{i+1} - x_i}{t_{i+1} - t_i} \cdot (t - t_i), \quad y(t) = y_i + \frac{y_{i+1} - y_i}{t_{i+1} - t_i} \cdot (t - t_i),$$

where $t_i \leq t \leq t_{i+1}$ and $1 \leq i < n$. In a geometric view, a trajectory in m -space after interpolation is equivalent to an $m+1$ -dimensional polygonal curve, because the interpolation forms straight line segments between sample points.

After interpolation, we can define distance at any t in $[t_1, t_n]$ instead of merely time stamps of sample points. This is necessary for weak simplification, since it could involve new points at any t . In order to bound the errors, the synchronized distance is used:

$$d(S, T) = \max_{t \in [t_1, t_n]} \|S(t) - T(t)\|. \quad (4)$$

In other words, the distance between two trajectories S and T is determined by the maximum distance between points corresponding to same time (see Figure 2). Given an error tolerance ϵ , the distance

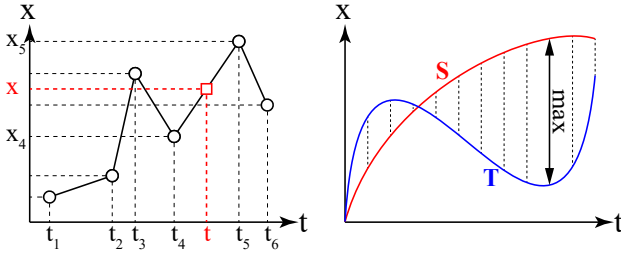


Figure 1: Interpolation

Figure 2: Sync-distance

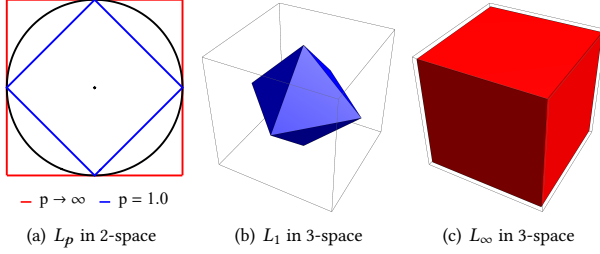


Figure 3: Distance metrics

between any two points at any time will be no greater than ϵ if $d(S, T) \leq \epsilon$.

All distance metrics can be used in Equation 4, though existing work on trajectory simplification is limited to the L_2 distance. Here we consider all metrics. The synchronized distance with respect to any L_p is

$$d(S, T) = \max_{t \in [t_1, t_n]} \|S(t) - T(t)\|_p$$

$$= \max_{t \in [t_1, t_n]} \left(\sum_{i=1}^m |S_i(t) - T_i(t)|^p \right)^{\frac{1}{p}}. \quad (5)$$

For any two points in space, the larger the value of p , the smaller the corresponding distance. Since all distances over time are smaller than the tolerance ϵ , the larger values of p lead to looser bounds. As a result, the synchronized distance using L_∞ admits more simplification points, as shown in Figure 3. Furthermore, L_∞ is a good approximation of other metrics, because $\|x\|_\infty \leq \|x\|_p \leq c \cdot \|x\|_\infty$, where $c = \sqrt[p]{m}$ is a constant¹. For example, the synchronized L_2 distance to the original trajectory will be no greater than $\sqrt{2} \cdot \epsilon$ if the synchronized L_∞ distance is bounded by ϵ in 2-space, and $\sqrt{3} \cdot \epsilon$ in 3-space. Another interesting property is that all metrics are equivalent in 1-space. Furthermore, L_1 is equivalent to L_∞ in 2-space after a linear transformation, so simplification under L_∞ is applicable to L_1 in 2-space. Note this does not hold true in higher dimensions, e.g., the unit sphere of L_1 , an octahedron (Figure 3(b)), is dual but not equivalent to that of L_∞ , a cube (Figure 3(c)), in 3-space.

Finally, we present the formal definition of the problem. Trajectory simplification is to reduce the number of points of the trajectory, after which the distance between input and output is bounded, so that data is compressed while the quality is still guaranteed.

PROBLEM DEFINITION. Given an error bound ϵ and an input trajectory in m -space T , we seek a trajectory S which consists of fewer points, and that the error between S and T is bounded by ϵ , i.e., $d(S, T) = \max_t \|S(t) - T(t)\|_\infty \leq \epsilon$.

The performance of simplification algorithms is evaluated by the compression ratio—the ratio of the input size to the output size. Under the normal representation, the size of the data depends on the length of the trajectory, so the compression ratio is $|T|/|S|$. Given the same error tolerance and input data, an algorithm with a higher compression ratio is more effective.

3 ALGORITHM DESIGN

In this section, we introduce LiMITS in detail. We first review the problem in 1-space and then discuss the generalization of existing algorithms to m -space. Next, we introduce multidimensional interpolation, which integrates information from all dimensions to construct simplified trajectories in m -space. Finally, we present a compact representation to further reduce the storage cost.

3.1 Simplification in One Dimension

We first review algorithms for trajectories in 1-space. One single number is enough to specify a location in 1-space, so trajectories are real functions of time, as well as polygonal curves in 2-space. For example, consider Figures 4(a) and 4(e), where X and Y are two trajectories in 1-space and both consist of six sample points.

3.1.1 Classic Methods. RDP [10] is a well-known algorithm for polygonal curve simplification. It recursively subdivides the curve into pieces until one line segment is sufficient to simplify each piece. For example, consider the trajectory X in Figures 4(a)–4(d). Initially, the algorithm keeps the first and last point p_1 and p_6 , and then finds the point that is vertically farthest from the line segment $\overline{p_1 p_6}$, which is p_3 . The vertical distance between p_3 and $\overline{p_1 p_6}$ is greater than the error tolerance ϵ so p_3 is kept. Next, the algorithm subdivides X into $\overline{p_1 p_2 p_3}$ and $\overline{p_3 p_4 p_5 p_6}$, and recursively call itself with each piece. Note that the line segment is sufficient to simplify the corresponding piece if the deviation at the farthest point is within ϵ , and hence the recursion terminates in this case. Eventually, all points in X are kept and no simplification takes place here. On the other hand, RDP is able to simplify the sample trajectory Y with three points $\overline{p_1 p_2 p_6}$ (see Figures 4(e) and 4(f)).

RDP produces a subset of the input as the result, so it belongs to strong simplification. It is easy to generalize RDP to high dimensions by simply changing the distance function. In Section 5, we review other algorithms that can be easily extended to high dimensions. According to the existing experimental studies [21, 45] and results in Section 4, these algorithms are not as effective as those algorithms based on sector intersection and link distance as described in Section 3.1.2 and 3.1.3.

3.1.2 Sector Intersection Methods. Effective strong simplification algorithms like OPW [26], OPT [15] simplify trajectories in 1-space through sector intersection. The greedy algorithm OPW checks input points one by one to determine if any can be omitted. As shown in Figures 5(a)–5(f), for the trajectory X , OPW first keeps the starting point p_1 , then tries to omit p_2 . However, the deviation at t_2 will exceed ϵ by replacing $\overline{p_1 p_2 p_3}$ with $\overline{p_1 p_3}$ so p_2 can not be

¹<https://math.stackexchange.com/questions/218046>

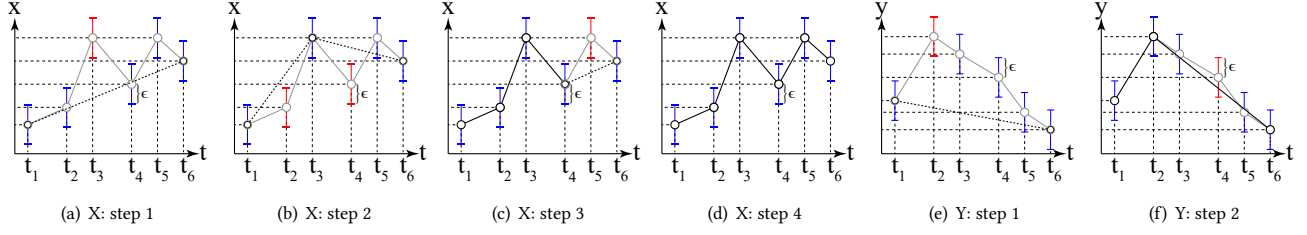


Figure 4: RDP algorithm

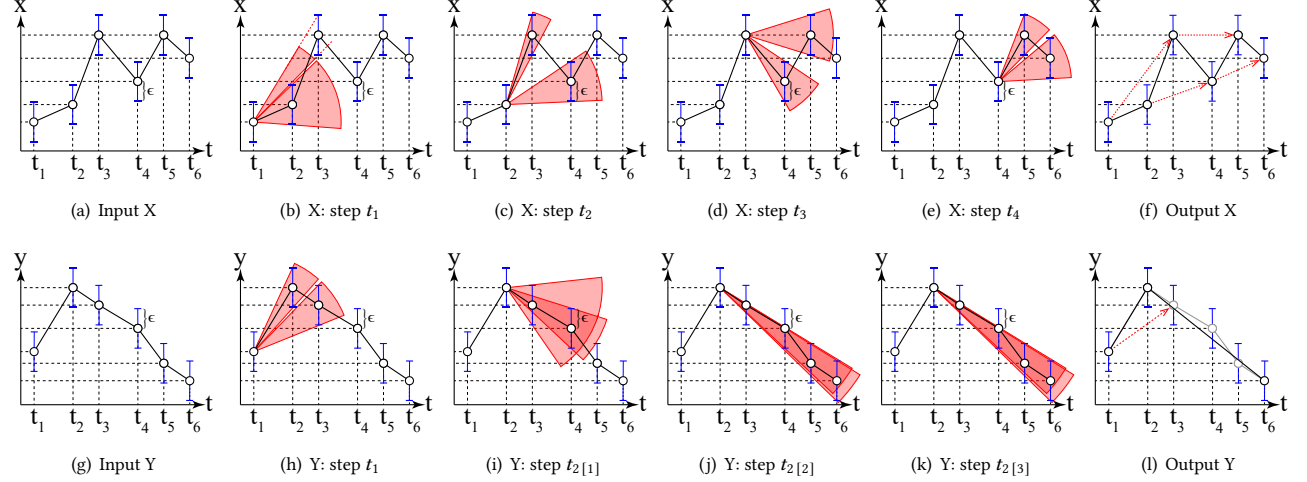


Figure 5: OPW algorithm

omitted. In the next iteration, OPW outputs p_2 and checks if p_3 can be omitted. Since the difference between $\overline{p_2 p_3 p_4}$ and $\overline{p_2 p_4}$ at t_3 is again greater than ϵ , p_3 should be kept as well. This process goes on until reaching the last point p_6 . Finally, the output is the same as the input and no simplification has taken place. On the other hand, OPW is able to simplify Y with three points, as shown in Figure 5(l).

The time complexity of OPW is $O(n^2)$ when a linear scan is used to compute the difference between $\overline{p_i p_{i+1} \dots p_{i+k}}$ and $\overline{p_i p_{i+k}}$. In [38], the time complexity of OPW is reduced to $O(n)$ through sector intersection. According to the problem definition, any output line $\overline{p_i p_{i+k}}$ must fall between $p_i l_j$ and $p_i h_j$, i.e., in the angle $\angle h_j p_i l_j$, where $i < j < i+k$, $l_j = (x_j - \epsilon, t_j)$ and $h_j = (x_j + \epsilon, t_j)$. Thus, $\overline{p_i p_{i+k}}$ must fall in the intersection of $\angle h_{i+1} p_i l_{i+1}, \dots, \angle h_{i+k} p_i l_{i+k}$. The intersections can be updated on the fly, and the current point can be omitted if and only if the intersection is not empty. For example, the intersections are all empty for X while only one is empty for Y , as shown in Figure 5, so X remains the same while Y only has three points after simplification. The overall time complexity becomes $O(n)$ as linear scans are no longer needed to determine if a point is to be omitted. OPW has optimal time complexity of $O(n)$. However, being a greedy algorithm, it does not eliminate many points.

OPT [15] is an optimal algorithm for strong simplification, which minimizes the size of the output. It first computes the visibility between any point p_i and p_j , i.e., whether the difference between

$\overline{p_i p_{i+1} \dots p_j}$ and $\overline{p_i p_j}$ is less than ϵ . Next, it applies dynamic programming on the visibility graph to construct an optimal path in terms of the number of points. Similar to OPW, the time complexity of OPT can also be reduced via sector intersection (from the original complexity of $O(n^3)$ to $O(n^2)$).

The algorithms mentioned above are strong simplifications and thus their performance is limited. Although CI [21] introduces a weak simplification algorithm based on sector intersection, it has the constraint that the time stamps of output trajectories should be from the input. Moreover, CI cannot omit points when intersections are empty, and thus cannot simplify trajectories like X as well. Sector intersection becomes complicated in higher dimensional spaces, for which we will introduce the generalization in Section 3.1.4. Compared with sector intersection, algorithms based on link distance discussed in the next subsection are more effective.

3.1.3 Link Distance Methods. The optimal simplification in 1-space is reducible to the minimum link path problem in simple polygons [11, 40]. Given an error tolerance ϵ , let $X_h(t) = X(t) + \epsilon$ and $X_l(t) = X(t) - \epsilon$ be two auxiliary trajectories (see Figure 6(a)). If a simplified trajectory $Z(t)$ falls between $X_h(t)$ and $X_l(t)$, then $X(t) - \epsilon \leq Z(t) \leq X(t) + \epsilon$ for any $t \in [t_1, t_n]$, and Z is a valid solution. The two auxiliary trajectories constitute a simple polygon after we connect their starting and ending points. The polygon is a boundary of feasible trajectories, because any point p outside the boundary results in the synchronized distance being greater than ϵ (see Figures 6(a) and 6(d)).

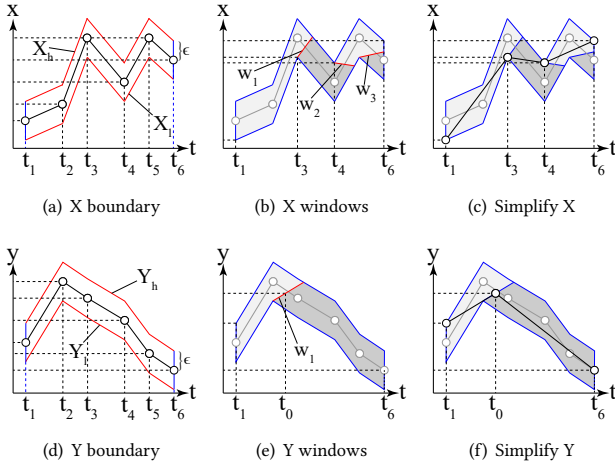


Figure 6: Link distance algorithm

Among all feasible trajectories inside the simple polygon, we seek the ones with minimum size, i.e., the fewest vertices. As a result, the trajectory simplification problem in 1-space is reduced to the problem of computing the *link distance* inside a simple polygon. The link distance between points or chords inside a polygon is defined to be the number of edges of a polyline inside the polygon connecting them. For example, the link distance between any points in a convex polygon is always one. The distance as well as the desired polyline in a polygon with n vertices can be computed in $O(n)$ time through a computational geometry algorithm [40]. In this algorithm, the polygon is subdivided into pieces by chords called *windows*, while the link distance from the source to the points in the same piece is constant. For example, consider X in Figure 6. Starting from the source vertical edge at t_1 , we construct three windows w_1 , w_2 and w_3 and subdivide the polygon into four parts until reaching the destination vertical edge at t_6 (see Figure 6(b)). In order to reach the rightmost edge, we have to go through at least two windows (i.e., w_1 and w_2), so two intermediate points are necessary to connect the leftmost and rightmost edges. Consequently, there are at least four points in the minimum link path, including the first and last points. Finally, we trace the windows back to the source to construct the optimal path with only four points (see Figure 6(c)). The optimal simplification of Y still consists of three points, as shown in Figure 6(f). Since it takes linear time to construct the simple polygon, the time complexity of optimal weak simplification in 1-space is $O(n)$.

Note that the algorithm in [40] uses polygon triangulation as its first step. Furthermore, it may re-triangulate part of the polygon during the process, so it could be time-consuming if the triangulation algorithm is not linear or not practical [8]. Fortunately, all polygons involved in weak simplification are monotone polygons since the trajectories are monotone in the temporal dimension, and hence they can be triangulated efficiently [42].

To sum up, the optimal weak simplification in 1-space is computable because we can compute the minimum link path in 2-space in $O(n)$ time. However, the link distance problem in 3-space has been proved to be NP-hard [17]. Consequently, the generalization

to high dimensions is not trivial. In Section 3.2, we propose an incremental approach to generalize link path construction to high dimensional space, namely multidimensional interpolation.

3.1.4 Generalization to High Dimensions. On the basis of existing algorithms in 1-space, we discuss their generalizations to high dimensions under L_∞ . In order to simplify the explanation, we illustrate the algorithms with example trajectory data in 2-space. For example, consider Figure 7(a), where the input trajectory consists of six sample points in 2-space, whose x-coordinate and y-coordinate values are X and Y in Figures 4 and 5, respectively.

As mentioned in Section 3.1.1, the generalization of algorithms such as RDP is straightforward, while it takes some effort to generalize sector intersection algorithms in Section 3.1.2. It is simple to perform sector intersection in 1-space, because the sectors are equivalent to vertical line segments—the unit ball of all metrics in 1-space. However, the unit balls become complicated in high dimensions. For example, it takes $\Theta(n \log n)$ time to compute the intersection of n unit balls of the L_p metric in 2-space or 3-space (e.g., disks or balls) and the complexity of the intersection by n unit balls in m -space could be up to $O(n^{\lfloor m/2 \rfloor})$ [25]. Consequently, sector intersection becomes difficult in high dimensional space. Fortunately, the unit balls of L_∞ are hypercubes, and it only takes $O(n)$ time to compute the intersection of n hypercubes in any space. Therefore, all algorithms based on sector intersection can be perfectly generalized to high dimensional space under the L_∞ metric.

The performance of the generalized algorithms is limited by their performance in 1-space. For algorithms like RDP, the L_∞ distance over all dimensions is less than ϵ if and only if the distance in each dimension is less than ϵ . For those algorithms based on sector intersection, the intersection is not empty if and only if all intersections in lower dimensions are not empty. Consequently, if the projection of a point in T_i is kept by the original algorithm in 1-space, then the point must be kept by the generalized algorithm, too. Thus, $|G(T)| \geq \max_{i=1}^m |g(T_i)|$, where $G(\cdot)$ is the generalized simplification algorithm in m -space while $g(\cdot)$ is its original version in 1-space. For example, the algorithms discussed in Sections 3.1.1 and 3.1.2 cannot simplify the input trajectory in Figure 7(a), because the simplification of one of its projections X (see Figures 4 and 5) consists of all points. In order to achieve a higher compression ratio, we resort to the generalization of optimal weak simplification based on link distance.

L_∞ also makes it possible to generalize the algorithm in Section 3.1.3, which is based on link distance, to higher dimensions. Here we illustrate the idea from the perspective of intrinsic properties of distance metrics. The synchronized distance with L_∞ is

$$\begin{aligned} d(S, T) &= \max_{t \in [t_1, t_n]} \|S(t) - T(t)\|_\infty \\ &= \max_{t \in [t_1, t_n]} \max_{i=1, 2, \dots, m} |S_i(t) - T_i(t)|. \end{aligned}$$

In this case, we can swap the maximum operators without changing the result, and hence the overall error under L_∞ is determined by the error in each dimension:

$$\begin{aligned} d(S, T) &= \max_{i=1}^m \max_{t \in [t_1, t_n]} |S_i(t) - T_i(t)| \\ &= \max_{i=1}^m d(S_i, T_i) \end{aligned} \quad (6)$$

where $d(S_i, T_i)$ is the synchronized distance in 1-space. It is therefore possible to break down the problem into simplification in 1-space and then combine the results to form a solution. On the contrary, for any finite p , the synchronized L_p distance could be greater than ϵ even if synchronized distances in all dimensions are bounded by ϵ . On the basis of this property, we propose the new approach, namely multidimensional interpolation, to generalize the algorithm in Section 3.1.3.

3.2 Multidimensional Interpolation

In this section, we first present the three implementations of LiMITS, including Direct Interpolation (DI), Midpoint Correlated Interpolation (MCI) and Various Interpolation (VI), which are all based on optimal simplification in 1-space introduced in Section 3.1.3. Then we introduce a compact representation of the output of LiMITS to further improve the compression ratio.

3.2.1 Direct interpolation. The first implementation is Direct Interpolation (DI). It directly combines results from all dimensions. Given a trajectory T in m -space, we first project it onto the planes spanned by each spatial dimension and the temporal dimension. Let the projection in the i -th plane be T_i , where the j -th sample point in T_i is $\langle T_i(T.t_j), T.t_j \rangle$. Then we simplify all projected trajectories in 1-space, after which we obtain m simplifications S_1, S_2, \dots, S_m . Finally, we combine all S_i by interpolation. Given any time t , the point in the output trajectory S is $\langle S_1(t), S_2(t), \dots, S_m(t) \rangle$. Since $T_i(t) - \epsilon \leq S_i(t) \leq T_i(t) + \epsilon$ for all i , the L_∞ distance between $T(t)$ and $S(t)$ should be within ϵ , so it is a feasible simplification. In practice, we only need to store points at time stamps of points from m simplifications and other points can be obtained through linear interpolation, as introduced in Section 2.

Algorithm 1: Direct Interpolation (DI)

```

input :input trajectory  $T$  and tolerance  $\epsilon$ 
output:output trajectory  $S$ 
1 for  $i \leftarrow 1$  to  $m$  do
2    $T_i \leftarrow T.\text{project}(\mathcal{P}_i)$ ;
3    $P_{T_i} \leftarrow \text{polygon}(T_i, \epsilon)$ ;
4    $R_i \leftarrow P_{T_i}.\text{link}(T_i(T.t_1) \pm \epsilon, T_i(T.t_n) \pm \epsilon)$ ;
5   if  $i = 1$  then  $S_1 \leftarrow R_i$ ;
6   else  $S_i \leftarrow \text{interpolation}(S_{i-1}, R_i)$ ;
7 end
8 return  $S_m$ ;

```

For the input trajectory in Figure 7(a), the projections are X and Y respectively (see Figures 7(b) and 7(c)). According to the computation in Section 3.1.3, the simplification of X consists of four points at t_1, t_3, t_4 and t_6 , while that of Y only consists of three points at t_1, t_0 and t_6 , as shown in Figure 7(d). Afterwards, one interpolation point is inserted in X at t_0 while two are inserted in Y at t_3 and t_4 , as Figure 7(e) shows. Finally, we take the x-coordinate and y-coordinate values from simplified X and Y at all time stamps to construct a simplification containing five points (see Figures 7(f) and 7(g)).

The pseudo-code is given by Algorithm 1. In each iteration, the algorithm first projects the input trajectory in the plane \mathcal{P}_i spanned

by the i -th spatial dimension and the temporal dimension (line 2) and then constructs a polygon with respect to the projection T_i (line 3). Next, the polygon along with the vertical edges at t_1 and t_n is the input of the link distance algorithm in 1-space, which computes the link distance between two edges $T_i(T.t_1) \pm \epsilon$ and $T_i(T.t_n) \pm \epsilon$, where $p \pm \epsilon$ represents the vertical line segment of length 2ϵ centered at a point p . The algorithm generates a feasible trajectory R_i for every T_i (line 4). Moreover, it integrates S_{i-1} (the combination of R_1, R_2, \dots, R_{i-1}) with R_i through interpolation and produces a trajectory S_i in i -space (line 6). The steps in each iteration take $O(n)$ time, so the time complexity is $O(nm)$. In practice, the dimension m is a constant and the algorithm is linear.

Algorithm 2: Midpoint Correlated Interpolation (MCI)

```

input :input trajectory  $T$  and tolerance  $\epsilon$ 
output:output trajectory  $S$ 
1 for  $i \leftarrow 1$  to  $m$  do
2    $T_i \leftarrow T.\text{project}(\mathcal{P}_i)$ ;
3   if  $i = 1$  then
4      $P_{T_i} \leftarrow \text{polygon}(T_i, \epsilon)$ ;
5      $S_1 \leftarrow P_{T_i}.\text{link}(T_1(T.t_1) \pm \epsilon, T_1(T.t_n) \pm \epsilon)$ 
6   else
7      $R_i \leftarrow \emptyset$ ;
8      $k \leftarrow T_i(T.t_1) \pm \epsilon$ ; // first  $k$  is an edge
9     for  $j \leftarrow 2$  to  $|S_{i-1}|$  do
10       $T'_j \leftarrow T_i \cap [S_{i-1}.t_{j-1}, S_{i-1}.t_j]$ ;
11       $P_{T'_j} \leftarrow \text{polygon}(T'_j, \epsilon)$ ;
12       $S'_j \leftarrow P_{T'_j}.\text{link}(k, T'_j(S_{i-1}.t_j) \pm \epsilon)$ ;
13       $R_i \leftarrow R_i \cup S'_j$ ;
14       $k \leftarrow S'_j(S_{i-1}.t_j)$ ; // other  $k$  are points
15   end
16    $S_i \leftarrow \text{interpolation}(S_{i-1}, R_i)$ ;
17 end
18 end
19 return  $S_m$ ;

```

3.2.2 Correlated Interpolation. DI is based on optimal simplification that is more effective than any 1-dimensional algorithm $g(\cdot)$ in Section 3.1.4, so it could produce better results than existing algorithms. However, it simply gathers information from all dimensions and ignores the correlation among the projections, so it is still ineffective. If two points from different projections coincide in time, only one point should be generated in the output, e.g., at the starting and ending times. Nonetheless, DI will still output two points even if their time stamps are very close. For example, the second and third points in the simplified trajectory in Figure 7(g) are redundant, because the second point in simplified X and the second point in simplified Y are close in time. In general, the size of the output could be up to $n \cdot m$ when ϵ is small. Consequently, further improvement is necessary.

Taking into account such correlation among projections, we propose the second implementation, namely Midpoint Correlated Interpolation (MCI), as shown in Algorithm 2. Similar to the first

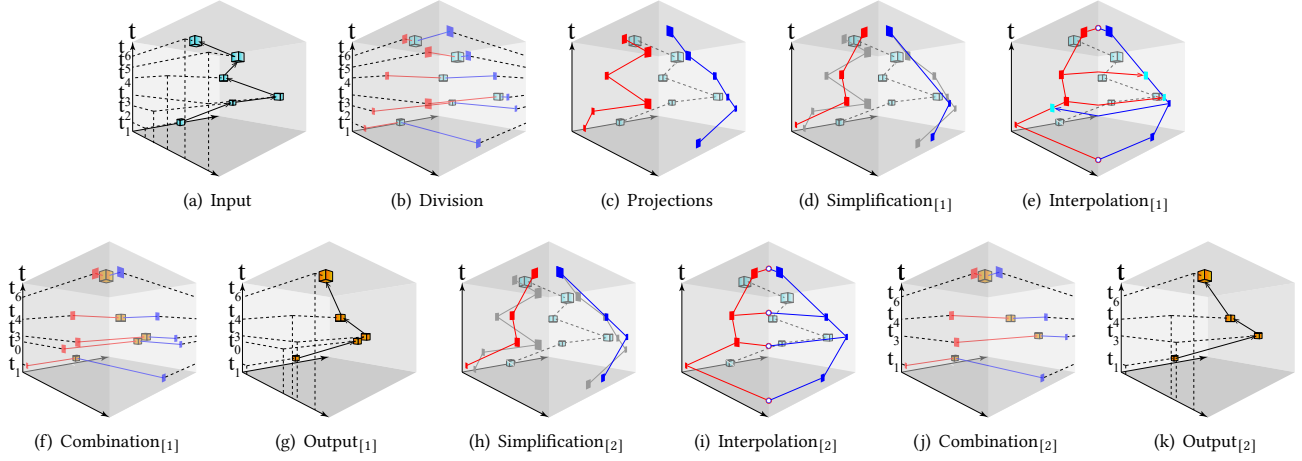


Figure 7: Multidimensional interpolation

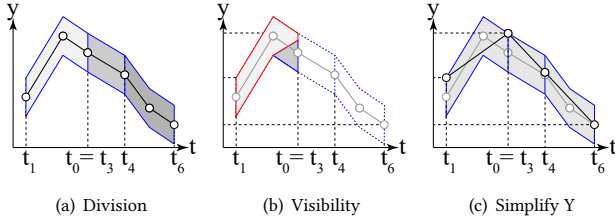


Figure 8: Correlated simplification

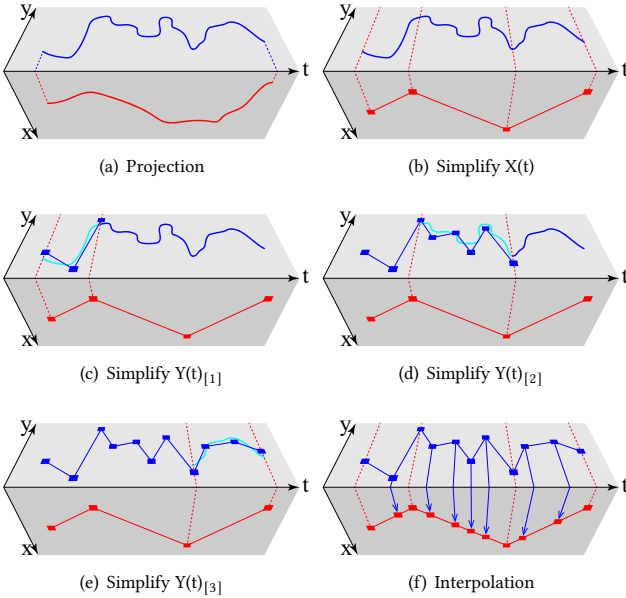


Figure 9: Correlated interpolation

implementation, the algorithm also generates m simplifications in the planes spanned by each spatial dimension and the temporal dimension. It produces a simplification for the projection of the

input trajectory in the i -space spanned by the first i spatial dimensions in the i -th iteration. Eventually, the algorithm produces an output trajectory in m -space after $i = m$. In the first iteration, the algorithm is equivalent to the optimal algorithm in 1-space, and generates a trajectory S_1 for the projection T_1 (line 5). In the second iteration, it subdivides the projection T_2 into $|S_1| - 1$ pieces according to the time stamps of S_1 (line 10). Here the link distance algorithm in Section 3.1.3 is applied in the polygons subdivided by the time stamps of S_1 (lines 11-12). After that, the algorithm simplifies each piece separately and connects the results together to form R_2 , a simplification for T_2 (line 13). At the end of the iteration, it combines S_1 with R_2 to form the new trajectory S_2 . In the third iteration, it subdivides T_3 according to S_2 and then simplifies it with R_3 . By combining R_3 and S_2 it produces S_3 . The process continues until $i > m$, and the trajectory S_m is the simplification for T . Note that the 1-space simplification in Algorithm 2 is slightly different from that in Algorithm 1, because it involves the link distance between a point, i.e., k in line 12, and an edge, while only link distance between edges is used in Algorithm 1.

For the same input trajectory in Figure 7(a), the algorithm first simplifies X , and the result remains the same (Figure 6(c)). Next, it subdivides Y according to the time stamps of the simplified X , i.e., t_1, t_3, t_4 and t_6 , as shown in Figure 8(a). Although there is a window in the first polygon between t_1 and t_3 , i.e., the maximum link distance in the polygon could be two, the window intersects the vertical edge at t_3 , as shown in Figure 8(b). Hence the link distance between the source and destination is only one, i.e., the simplification of the first piece of Y only has two points. The simplification of the other two pieces is similar and the final results for Y are shown in Figures 8(c) and 7(h). Note that now the final result, i.e., the combination of simplified X and Y , only contains four points, though the simplification of Y is not optimal (see Figures 7(j) and 7(k)). Another example of a longer trajectory is shown in Figure 9. Note that we use curves to demonstrate polygonal curves containing lots of sample points in this example.

MCI outperforms DI because it exploits the correlation among projections. Algorithm 1 simplifies projections separately, while Algorithm 2 simplifies later projections on the basis of the current

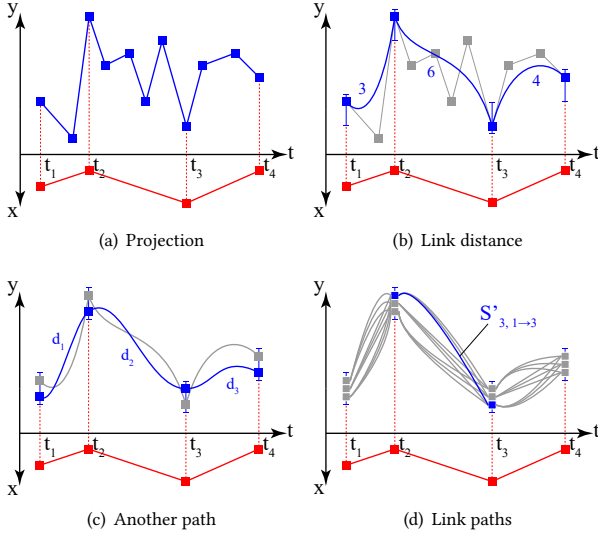


Figure 10: Various interpolation

simplification S_{i-1} . In Algorithm 1, we construct the simplified trajectory incrementally, and the trajectory S_i contains all points from S_{i-1} . No matter how we simplify T_i , the interpolation points at time stamps from S_{i-1} are always kept. Consequently, we construct the simplification of T_i according to the time stamps from S_{i-1} , and all interpolation points at these time stamps will not increase the size of S_i . For example, Y can be simplified with three points in Algorithm 1 while it is four points in Algorithm 2, but the result is even better because the extra point in Y and the third point in simplified X coincide (see Figure 7(i)). Furthermore, it makes the simplification of T_i easier, as the link distance between two consecutive time stamps could be very short. For example, all simplifications of Y pieces only consist of one line segment, as Figure 8(c) shows.

In the j -th piece of T_i , there could be multiple optimal simplifications and their last vertices form a continuous vertical line segment at t_{j+1} . In practice, we take the one whose last vertex is the midpoint of the line segment, and this is the reason why we call the algorithm MCI. The steps in the inner loop (lines 9-15) takes $O(n_j)$ time for the j -th piece of T_i whose size is n_j . In total, the sum of n_j is $O(n)$, so the time complexity of the second implementation is $O(mn)$ as well.

3.2.3 Various Interpolation. In the correlated interpolation algorithm, the simplified trajectory S'_j ends with the point k , which becomes the starting point of S'_{j+1} . The point k could be any one on the edge $T'_j(S_{i-1}.t_j) \pm \epsilon$ as long as the local link distance inside the polygon is minimized. In practice, we take the average position of all possible k , i.e., the midpoint. However, such greedy strategy to select k can be further improved and we should take into account all points on the vertical edges, including those resulting in non-optimal local link distance. We therefore select connecting points k through a better strategy to reduce the size of R_i as well as S_i . In this section, we propose the third implementation of LiMITS, namely Various Interpolation (VI).

Algorithm 3: Various Interpolation (VI)

```

input : input trajectory  $T$ ,
        tolerance  $\epsilon$  and sampling rate  $r$ 
output: output trajectory  $S$ 
1 for  $i \leftarrow 1$  to  $m$  do
2    $T_i \leftarrow T.\text{project}(\mathcal{P}_i)$ ;
3   if  $i = 1$  then
4      $P_{T_1} \leftarrow \text{polygon}(T_1, \epsilon)$ ;
5      $S_1 \leftarrow P_{T_1}.\text{link}(T_1(T.t_1) \pm \epsilon, T_1(T.t_n) \pm \epsilon)$ 
6   else
7     for  $q \leftarrow 0$  to  $r$  do  $v_{1,q} \leftarrow 1$ ;
8     for  $j \leftarrow 2$  to  $|S_{i-1}|$  do
9        $T'_j \leftarrow T_i \cap [S_{i-1}.t_{j-1}, S_{i-1}.t_j]$ ;
10       $P_{T'_j} \leftarrow \text{polygon}(T'_j, \epsilon)$ ;
11      for  $q \leftarrow 0$  to  $r$  do
12        for  $p \leftarrow 0$  to  $r$  do
13           $k_q \leftarrow T'_j(S_{i-1}.t_j) + (2q/r - 1) \cdot \epsilon$ ;
14           $k_p \leftarrow T'_j(S_{i-1}.t_{j-1}) + (2p/r - 1) \cdot \epsilon$ ;
15           $S'_{j,p \rightarrow q} \leftarrow P_{T'_j}.\text{link}(k_p, k_q)$ ;
16        end
17         $v_{j,q} \leftarrow \min_p \{v_{j-1,p} + |S'_{j,p \rightarrow q}| - 1\}$ ;
18      end
19    end
20     $R_i \leftarrow \emptyset$ ;  $q \leftarrow \text{argmin}_p v_{|S_{i-1}|,p}$ ;
21    for  $j \leftarrow |S_{i-1}|$  downto 2 do
22       $p \leftarrow \text{argmin}_p \{v_{j-1,p} + |S'_{j,p \rightarrow q}| - 1\}$ ;
23       $R_i \leftarrow R_i \cup S'_{j,p \rightarrow q}$ ;  $q \leftarrow p$ ;
24    end
25     $S_i \leftarrow \text{interpolation}(S_{i-1}, R_i)$ ;
26  end
27 end
28 return  $S_m$ ;

```

For the input trajectory in Figure 9, the simplification in the x dimension consists of four points, while the simplification in the y dimension consists of 11 points, as shown in Figure 10(a). More precisely, the projection in the y dimension is subdivided into three pieces, because the simplification in the x dimension has four points. When we apply the correlated interpolation, the link distances in the three pieces are three, six and four respectively (see Figure 10(b)). Nevertheless, a feasible simplification may pass through the vertical edges at any point (see Figure 10(c)), and the link distances could be d_1 , d_2 and d_3 respectively. Consequently, there are an infinite number of possible positions of k on each vertical edge. In order to select k effectively, we propose what we call Various Interpolation (VI). We first sample $r + 1$ uniformly distributed points on each vertical edge as candidates. Next, we construct $O(r^2)$ minimum link paths between candidates on starting and ending edges in each piece. After that, an optimal simplification through candidates can be computed via dynamic programming. Let $v_{j,q}$ be the minimum number of vertices to reach the q -th candidate at the j -th vertical

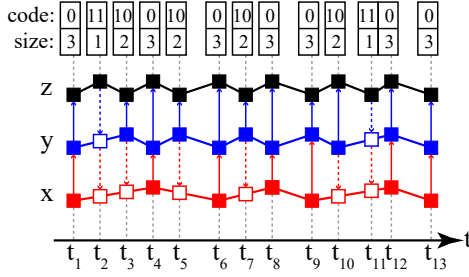


Figure 11: Compact representation

edge, which can be computed recursively:

$$v_{j,q} = \begin{cases} \min_p \{v_{j-1,p} + |S'_{j,p \rightarrow q}| - 1\} & j > 1, \\ 1 & j = 1, \end{cases} \quad (7)$$

where $|S'_{j,p \rightarrow q}|$ is the link distance between the candidates p and q on the consecutive vertical edges (see Figure 10(d)).

The pseudo-code is shown in Algorithm 3. Again, we subdivide the projection into pieces and construct corresponding polygons in each iteration. However, different from the correlated interpolation, we then sample candidates (lines 13-14) and construct link paths (line 15). Meanwhile, we compute $v_{j,q}$ according to Equation 7 (line 17). Next, we construct an optimal simplification R_i through candidates according to $v_{j,q}$ values (line 20-24). The time complexity of the algorithm is $O(r^2 mn)$. In Section 4, we show that a small value of r is sufficient to achieve better performance than the correlated interpolation. In practice, the link distance algorithm is only applied $r + 1$ times in each iteration of the inner loop and thus we do not need to perform the link distance computation between all $O(r^2)$ pairs. Furthermore, we do not need to store all link paths and only construct those that form the optimal simplification. Regardless, these minor improvements do not affect the compression ratio as well as the time complexity. For more details of the implementation, please refer to the source code available online (see Section 4).

3.2.4 Compact Representation. Besides the normal format where a trajectory is represented by series of points, LiMITS also allows a compact representation which leads to even better performance. Although there are no consecutive collinear points in the results after simplification, some points could still be collinear in one or more dimensions. For example, consider the output trajectory in Figure 11, where the first four points are collinear in the x dimension because the x-coordinate values of the second and third points are interpolation values. Similarly, the first three points are collinear in the y dimension. In the compact representation, we do not store these interpolation values so that the storage cost is reduced.

In LiMITS, an output point in m -space contains 0 to $m - 1$ interpolation values. Furthermore, the coordinate values of the first k dimensions are interpolation values while the remaining coordinate values are not, if the point contains k interpolation values. We therefore only store the coordinate values of the last $m - k$ dimensions. For example, we store all coordinate values of p_1 , only the z-coordinate value of p_2 and both the y-coordinate value and the z-coordinate value of p_3 for the trajectory in Figure 11. As a result, the number of coordinate values in a point varies in the compact

Table 1: Dataset information

Dataset	Source	Records	Points	Dimension
Wiener	Random	100	200,000	2D/3D
Bee	Animal	876	2,523,180	2D
Danio		42	64,586	3D
Beijing	Human	18,655	24,106,047	2D/3D
Mopsi		6,779	7,612,499	2D/3D

format while it is fixed in the normal format. Consequently, we append an extra codeword to each point to indicate the number of coordinate values, where the an entropy encoding like Huffman coding [13] may be used.

For example, 2-bit codewords are sufficient for trajectories in 3-space. In the experiments, we use the codewords 11, 10 and 0 to indicate one, two and three coordinate values respectively, because the frequencies of points containing all three coordinate values are the highest. The compact representation of the example trajectory in Figure 11 is $\langle 0x_1y_1z_1t_1, 11z_2t_2, 10y_3z_3t_3, \dots, 0x_{13}y_{13}z_{13}t_{13} \rangle$, where x_i, y_i, z_i and t_i are floating point numbers which require 64 bits if double precision is chosen. There are 7 points containing all three coordinate values, 4 points containing two and 2 points containing one, so the size of the compact representation is $7 \times (1 + 4 \times 64) + 4 \times (2 + 3 \times 64) + 2 \times (2 + 2 \times 64) = 2,835$ bits, while that of the normal representation is $13 \times 4 \times 64 = 3,328$ bits.

The conversion between compact and normal representations is lossless and only takes linear time. In particular, for each point, we first read the codeword to obtain its size, and then read the corresponding number of floating-point numbers. This process continues until we have decoded all points in the compact representation. After decoding all points, we compute the remaining coordinate values through interpolation. Note that the compact representation is not applicable to other conventional algorithms, because the colinearity in the individual dimensions is not guaranteed.

4 EXPERIMENTS

In this section, we compare LiMITS with existing algorithms using real and generated data, in terms of both effectiveness and efficiency. All algorithms are implemented in Java and run on a computer with Intel Xeon CPU E5-2620 @ 2.10GHz and 64GB memory. We select OPT [15] and CIS [21] as the representatives of strong simplification, and CIW [21] as the representative of weak simplification, because their compression ratios outperform other algorithms [21, 45]. RDP [10] is a classic algorithm for the problem so we also implemented it as a baseline. The source code is publicly available online.²

4.1 Datasets

In order to evaluate the performance of algorithms, we use following trajectory datasets in our experiments: Beijing (Geolife) [47], Mopsi [24], Bee [4], Danio [39] and Wiener [23]. The detailed information including the numbers of trajectories and sample points is given in Table 1.

The first two datasets consist of trajectories collected by human users. These human trajectories record wide ranges of activities,

²<https://github.com/yhhan19/LiMITS>

such as walking, hiking and traveling via vehicles. A sample point in the human trajectory data usually consists of a latitude, a longitude, an altitude and a time stamp. The majority of the human trajectories contain valid altitudes and hence are used as 3D data. We also use these trajectories as 2D data by ignoring the altitudes. The other two datasets from animals are collected from video data through object tracking techniques. The bees are on a planar bee hive so their trajectories are 2D, while the danios are in a fish tank so the trajectories are 3D. Besides the real datasets, we also generated random trajectories simulating Brownian motion according to the standard Wiener process [23].

4.2 Results

We first evaluate the effectiveness of the algorithms in terms of compression ratios. We plot the *relative compression ratios* with respect to the error tolerance ϵ in Figure 12. The relative compression ratio is defined as the ratio of the output size of an algorithm to that of the baseline algorithm RDP. For example, if RDP produces an output trajectory of 10 points while another algorithm produces only 7 points, the relative compression ratio of that algorithm is 70%. We observe similar patterns in Figure 12. First, the relative performance of the algorithms for the different datasets is almost the same. Second, The performance of weak simplification methods is better than that of strong simplification methods, i.e., RDP and CIS are not as effective as MCI and VI. Third, we find the relative compression ratios of various algorithms (except CIW) to be relatively independent to the error tolerance when the error tolerance is not small. Moreover, the compression ratio of CIW becomes close to CIS as the error tolerance increases, and it is eventually even worse than RDP and/or CIS in the Bee and Wiener datasets.

Among the existing algorithms, OPT and CIW have the highest compression ratios. Moreover, CIW is more effective when the error tolerance is small while OPT is more effective for great error tolerance. Regarding the implementations of LiMITS, we only plot MCI and VI because the DI is ineffective in practice. MCI usually outperforms other algorithms and VI is always the most effective. The compression ratio is further improved after we apply the compact format (see Figure 11). We observe that both MCI and VI are significantly more effective than all other algorithms after using the compact representation. Note that OPT has a quadratic time complexity, and hence it is time-consuming to evaluate OPT on the largest dataset Beijing. Consequently, we sample a small part (10%) and plot the results as dashed lines in Figures 12(a) and 12(b).

Next, we report the efficiency of the algorithms. We plot the execution time per sample point with respect to the error tolerance in Figure 14. Among all algorithms, CIS and CIW are the most efficient, because they are one-pass algorithms as they only access all sample points once. RDP has a time complexity of $O(n \log n)$ so it takes more time than CIS and CIW. The execution time of MCI is close to that of RDP though it is a linear algorithm. Because MCI involves complicated link distance computation, it has a greater constant factor. VI is $O(r^2 mn)$ where r is the sampling rate, m is the dimension of the space, and n is the number of vertices in the input trajectory. Consequently, VI takes even more time than MCI since it has a great constant factor. In fact, the dynamic programming in VI is very efficient. However, VI takes as $r + 1$ times as much time as

MCI, because the time is mainly spent on link distance computation and VI computes link distance $r + 1$ times in each piece. Here the sampling rate of candidates in VI is 10, and VI takes approximately 11 times as much time as MCI on average. The optimal strong simplification method OPT outperforms RDP and CIS in terms of the compression ratio but is most time consuming. On the other hand, VI has a higher compression ratio than OPT, but only takes 10% of its execution time on average.

We also demonstrate effects of the sampling rate of the candidates in VI. As Figure 15 shows, the greater sampling rate leads to better compression ratios, but also results in greater execution time. Moreover, the execution time increases linearly with the growth of r . However, the difference in terms of compression ratios is not much, i.e., within 3% in the Mopsi dataset. As a result, a moderate value of r is sufficient to achieve high compression ratios.

To sum up, VI has the highest compression ratio among all approaches even without the compact representation, but takes more time to simplify a trajectory. MCI is less effective but more efficient than VI, and it completely outperforms the state-of-the-art algorithms (i.e., OPT and CIW) if the compact representation is used. Different from OPT and CIW, MCI and VI have high compression ratios no matter how small or large the error tolerance is.

5 RELATED WORK

In this section, we review work closely related to the trajectory simplification problem, including data compression, error metrics, and existing algorithms exhibiting both strong and weak simplification.

5.1 Data compression

LiMITS is a weak simplification approach for trajectory data, and thus belongs to data compression techniques. Data compression can be either lossless or lossy. The original data can be perfectly restored after lossless compression [13, 44, 48], but its compression ratio is limited by the information entropy [36]. Lossless algorithms for trajectory data [9, 29] are not effective when compared with lossy algorithms [45], because trajectories are in the form of floating-point numbers. In other words, the alphabet size tends to be infinity when precision of data increases. Consequently, the statistical redundancy is very rare and thus lossless compression is ineffective, as shown in [11]. Lossy compression can achieve higher compression ratios by simply discarding unnecessary data, usually designed for images, audio and video [28, 37, 43]. The quality of the data is preserved after lossy compression, i.e., users can hardly distinguish between the original and compressed data. Trajectories are similar to audio/video since they are all time series, and the majority of trajectory simplification algorithms are lossy, too. However, different error metrics should be used to measure the data quality.

5.2 Error metrics

There are a variety of error metrics in trajectory simplification. The first metric is the perpendicular Euclidean distance. It measures the perpendicular distance between output points and their associated line segments in the input, and then takes the maximum of all these values as the result. The perpendicular distance considers the spatial information of the trajectory data, so a small distance indicates that the shapes of two trajectories are similar. However,

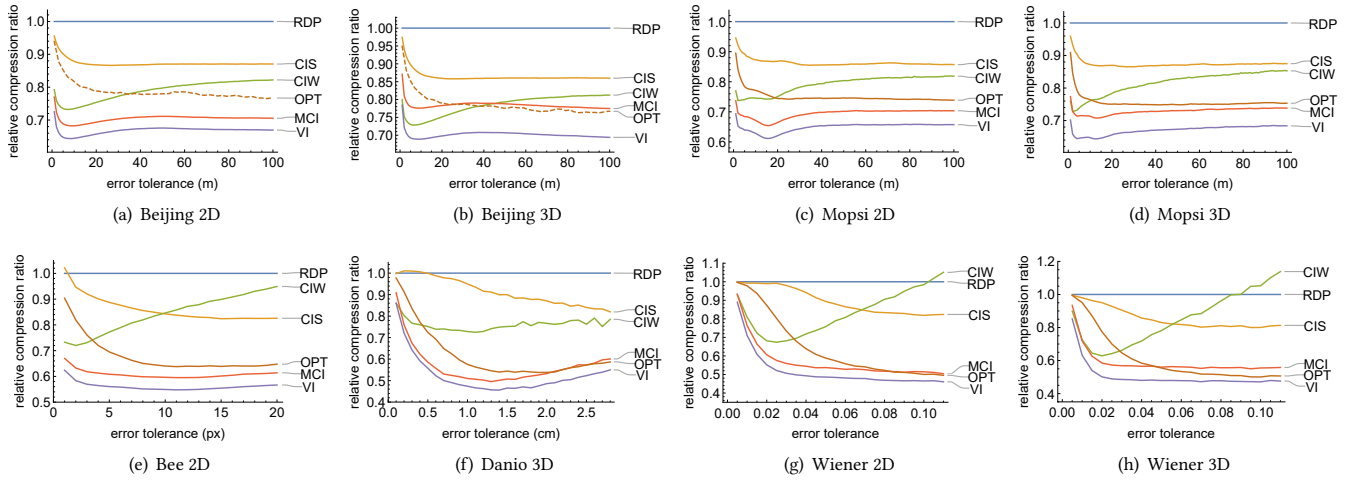


Figure 12: Relative compression ratio

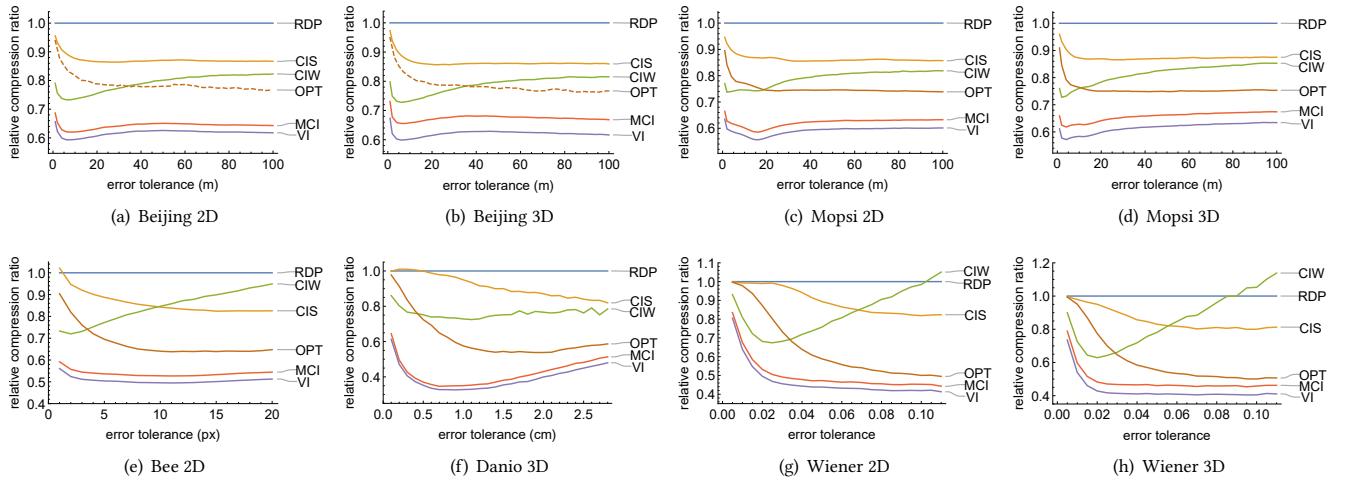


Figure 13: Compact representation

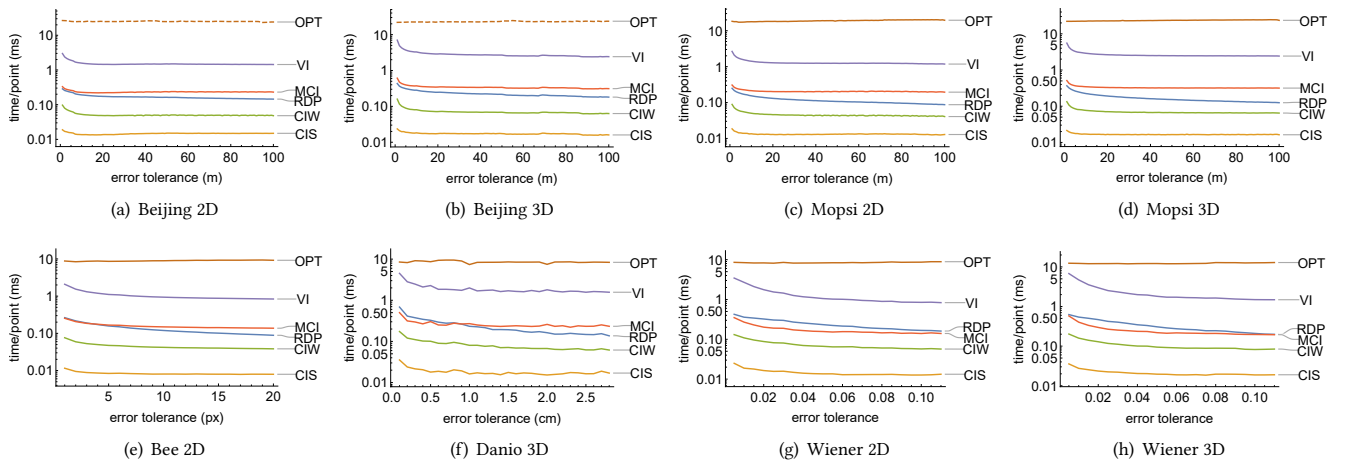


Figure 14: Execution time

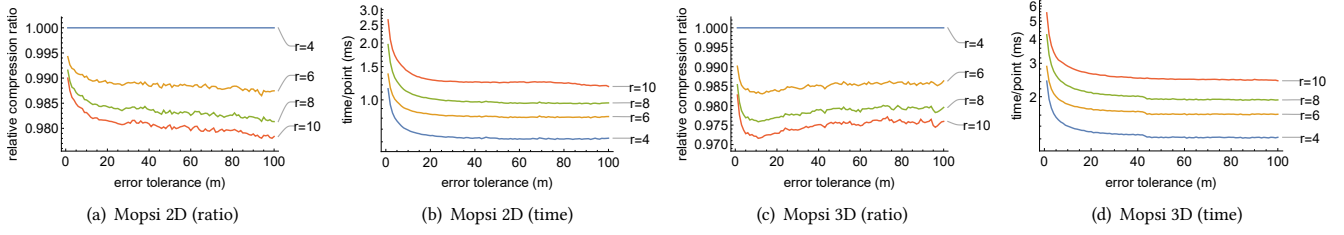


Figure 15: Various interpolation

temporal information is as important, and thus two trajectories with similar shape may be very different in many other aspects like the starting/ending times, velocity, and acceleration. As a result, many other approaches bound errors through use of synchronized Euclidean distance, a simplified version of the Fréchet distance, which is in the form of Equation 4 while using the L_2 metric. The synchronized distance takes into account both spatial and temporal information, and therefore the trajectories after simplification are similar in both space and time. Existing work uses the synchronized L_2 distance to measure the error. In this paper, we focus on using L_∞ , because the algorithms using this metric can be extended to higher dimensions. Besides the perpendicular and synchronized distances, other metrics based on direction [22] and velocity [20] are also useful to measure the similarity between trajectories.

5.3 Simplification algorithms

Existing trajectory simplification algorithms compatible with the synchronized L_2 distance include RDP [10], SQUISH [27], OPW [26], OPT [15] and CI [21], among which all except CI exhibit strong simplification. RDP is a well-known algorithm for trajectory simplification. The worst-case time complexity of RDP is $O(n^2)$, but can be reduced to $O(n \log n)$ and it is very efficient in practice. SQUISH maintains a priority queue of points, in which the priority of a point is given by the error introduced upon removing the point from the trajectories. Taking advantage of the priority queue, it can remove the point with the lowest priority efficiently, and takes $O(n \log n)$ time to simplify a trajectory with n points. OPW is a greedy algorithm that always tries to simplify as many points as possible with one line segment at each iteration. The time complexity of OPW is $O(n^2)$ but it can be reduced to $O(n)$ through sector intersection [21, 38]. OPT is an optimal algorithm in terms of the compression ratio for strong simplification [15], which has an $O(n^3)$ time complexity but is able to produce minimum subsets. Similar to OPW, the time cost of OPT can be reduced to $O(n^2)$ through sector intersection as well [11]. All these algorithms are strong simplification algorithms, and therefore the compression ratio is limited. The state-of-the-art approach is CI, which improves OPW via sector intersection. Moreover, it introduces both strong and weak simplification algorithms, namely CIS and CIW, respectively. Although CIW is a weak simplification, it still constricts the interpolation points in the temporal dimension, whereas there are no such constraints in LiMITS.

Besides strong and weak simplifications, we also decompose all algorithms into two groups according to their methodology.

Trajectory data, in the form of a series of sample points, are discrete geometric objects, so all algorithms involve a varying degrees of knowledge from computational geometry. More precisely, the only geometric operation used in RDP, SQUISH, and naive OPW/OPT is the evaluation of distances between points, while improved OPW/OPT and CI implement the more complicated sector intersection operation in order to achieve better performance. Consequently, it is simple to generalize algorithms like RDP to other metrics in high dimensions by changing the distance evaluation function, while other algorithms using sector intersection are difficult to generalize, because the intersection of hyper-balls with respect to L_p metrics in m -space takes more than linear time when $m > 1$ and $1 < p < \infty$. In 2-space, CI tackles L_2 by approximating circles with regular polygons of n vertices so that the computation of sector intersections is simplified, which is equivalent to L_∞ when $n = 4$. In this paper, we use L_∞ directly without any approximation.

In [45], the authors conducted a comprehensive experimental study on simplification algorithms invented in the last several decades, including other algorithms that do not use the synchronized distance and also those that do not use error tolerances. On the basis of the experimental study, the authors point out several directions worthy of further exploration. First, it is suggested to trade execution time and memory space for effective simplification. Second, it is useful to study new error metrics which are suitable for real applications. Finally, inter-trajectory redundancy has not been exploited yet, which could lead to even higher performance. In our work, we followed the first two directions. We first explored the metric L_∞ , making simplification in high dimensions feasible. Next, we proposed LiMITS, a linear but more complicated algorithm, to simplify trajectories more effectively.

6 CONCLUSION

In this paper, we explored the problem of trajectory simplification under the L_∞ metric, which makes it possible to generalize existing algorithms to higher dimensions. Next, we proposed three linear weak simplification algorithms based on multidimensional interpolation, namely LiMITS. It also allows a compact representation leading to an even higher compression ratio. Finally, we conducted extensive experiments on real datasets showing that LiMITS outperforms existing algorithms in terms of the compression ratio.

Notwithstanding, LiMITS is not an optimal algorithm, i.e., the number of points in the output is not minimized. Although the optimal weak simplification is reducible to the link distance problem, link distance computation in high dimensions is still an open problem [17]. Nonetheless, the polytopes in the trajectory simplification

problem are not arbitrary but tube-shaped. As a result, it is interesting to investigate whether there is an efficient algorithm to compute link distance inside the tube-shaped polytopes, and then simplify trajectories with the minimum link paths. Another direction for future work is exploring the relationship between trajectories that we obtain with those generated as a result of performing spatial network queries (e.g., [32–34]) and trajectory compression in networks [6, 11, 16, 19, 38] as a great proportion of trajectories is from spatial networks.

REFERENCES

- [1] Helmut Alt and Micheal Godau. 1995. Computing the Fréchet distance between two polygonal curves. *International Journal of Computational Geometry & Applications* 05, 01n02 (1995), 75–91.
- [2] Arnon Amir, Alon Efrat, Piotr Indyk, and Hanan Samet. 1999. Efficient regular data structures and algorithms for location and proximity problems. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*. IEEE, 160–170.
- [3] Chuan-Heng Ang, Hanan Samet, and Clifford A. Shaffer. 1990. A new region expansion for quadrees. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 12, 7 (1990), 682–686.
- [4] Katarzyna Bozek, Laetitia Hebert, Alexander S. Mikheyev, and Greg J. Stephens. 2018. Towards dense object tracking in a 2D honeybee hive. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 4185–4193.
- [5] Karl Bringmann, Marvin Künnemann, and André Nusser. 2019. Walking the dog fast in practice: algorithm engineering of the Fréchet distance. In *Proceedings of the 35th International Symposium on Computational Geometry*, Vol. 129. LIPIcs, 17:1–17:21.
- [6] Hu Cao and Ouri Wolfson. 2005. Nonmaterialized motion information in transport networks. In *International Conference on Database Theory*. Springer, 173–188.
- [7] Hu Cao, Ouri Wolfson, and Goce Trajcevski. 2006. Spatio-temporal data reduction with deterministic error bounds. *The VLDB Journal* 15, 3 (2006), 211–228.
- [8] Bernard Chazelle. 1991. Triangulating a simple polygon in linear time. *Discrete & Computational Geometry* 6, 3 (1991), 485–524.
- [9] Philippe Cudre-Mauroux, Eugene Wu, and Samuel Madden. 2010. Trajstore: an adaptive storage system for very large trajectory data sets. In *26th International Conference on Data Engineering*. IEEE, 109–120.
- [10] David H. Douglas and Thomas K. Peucker. 1973. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization* 10, 2 (1973), 112–122.
- [11] Yunheng Han, Weiwei Sun, and Baihua Zheng. 2017. COMPRESS: a comprehensive framework of trajectory compression in road networks. *ACM Transactions on Database Systems* 42, 2 (2017), 11.
- [12] Imai Hiroshi and Iri Masao. 1988. Polygonal approximations of a curve—formulations and algorithms. In *Machine Intelligence and Pattern Recognition*. Vol. 6. Elsevier, 71–86.
- [13] David A. Huffman. 1952. A method for the construction of minimum-redundancy codes. *Proceedings of the IRE* 40, 9 (1952), 1098–1101.
- [14] Insung Ihm and Bruce Naylor. 1991. Piecewise linear approximations of digitized space curves with applications. In *Scientific Visualization of Physical Phenomena*. Springer, 545–569.
- [15] Hiroshi Imai and Masao Iri. 1986. Computational-geometric methods for polygonal approximations of a curve. *Computer Vision, Graphics, and Image Processing* 36, 1 (1986), 31–41.
- [16] Georgios Kellaris, Nikos Pelekis, and Yannis Theodoridis. 2013. Map-matched trajectory compression. *Journal of Systems and Software* 86, 6 (2013), 1566–1579.
- [17] Irina Kostitsyna, Maarten Löffler, Frank Staals, and Valentin Polishchuk. 2016. On the complexity of minimum-link path problems. In *Proceedings of the 32th International Symposium on Computational Geometry*, Vol. 51. LIPIcs, 49:1–49:16.
- [18] André Langevin and Diane Riopel. 2005. *Logistics Systems: Design and Optimization*. Springer.
- [19] Tianyi Li, Ruikai Huang, Lu Chen, Christian S. Jensen, and Torben Bach Pedersen. 2020. Compression of uncertain trajectories in road networks. In *Proceedings of the VLDB Endowment*, Vol. 13. VLDB Endowment, 1050–1063.
- [20] Chih-Yu Lin, Chih-Chieh Hung, and Po-Ruey Lei. 2016. A velocity-preserving trajectory simplification approach. In *2016 Conference on Technologies and Applications of Artificial Intelligence (TAAI)*. IEEE, 58–65.
- [21] Xuelian Lin, Jiahao Jiang, Shuai Ma, Yimeng Zuo, and Chunming Hu. 2019. One-pass trajectory simplification using the synchronous Euclidean distance. *The VLDB Journal* 28, 6 (2019), 897–921.
- [22] Cheng Long, Raymond Chi-Wing Wong, and H. V. Jagadish. 2013. Direction-preserving trajectory simplification. In *Proceedings of the VLDB Endowment*, Vol. 6. VLDB Endowment, 949–960.
- [23] Vidyadhar Mandrekar. 1995. Mathematical work of Norbert Wiener. *Notices of the AMS* 42, 6 (1995), 664–669.
- [24] Radu Marinescu-Istodor and Pasi Fränti. 2017. Grid-based method for GPS route analysis for retrieval. *ACM Transactions on Spatial Algorithms and Systems* 3, 3, Article 8 (Sept. 2017), 28 pages.
- [25] Peter McMullen. 1970. The maximum numbers of faces of a convex polytope. *Mathematika* 17, 2 (1970), 179–184.
- [26] Nirvana Meratnia and Rolf A. de By. 2004. Spatiotemporal compression techniques for moving point objects. In *International Conference on Extending Database Technology*. Springer, 765–782.
- [27] Jonathan Muckell, Paul W. Olsen, Jeong-Hyon Hwang, Catherine T. Lawson, and S. S. Ravi. 2014. Compression of trajectory data: a comprehensive evaluation and new approach. *Geoinformatica* 18, 3 (01 Jul 2014), 435–460.
- [28] Hans Georg Musmann. 2006. Genesis of the MP3 audio coding standard. *IEEE Transactions on Consumer Electronics* 52, 3 (2006), 1043–1049.
- [29] Aiden Nibali and Zhen He. 2015. Trajic: An effective compression system for trajectory data. *IEEE Transactions on Knowledge and Data Engineering* 27, 11 (2015), 3138–3151.
- [30] Hanan Samet. 1983. A quadtree medial axis transform. *Commun. ACM* 26, 9 (1983), 680–693.
- [31] Hanan Samet. 1985. Reconstruction of quadrees from quadtree medial axis transforms. *Computer Vision, Graphics, and Image Processing* 29, 3 (March 1985), 311–328. Also University of Maryland Computer Science Technical Report TR-1224, October 1982.
- [32] Jagan Sankaranarayanan, Houman Alborzi, and Hanan Samet. 2006. Distance join queries on spatial networks. In *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*. ACM, 211–218.
- [33] Jagan Sankaranarayanan and Hanan Samet. 2010. Query processing using distance oracles for spatial networks. *IEEE Transactions on Knowledge and Data Engineering* 22, 8 (2010), 1158–1175.
- [34] Jagan Sankaranarayanan, Hanan Samet, and Houman Alborzi. 2009. Path oracles for spatial networks. In *Proceedings of the VLDB Endowment*, Vol. 2. VLDB Endowment, 1210–1221.
- [35] Charles L. Seitz and California Institute of Technology. 1989. *Advanced research in VLSI: Proceedings of the Decennial Caltech Conference on VLSI, March 1989*. MIT Press.
- [36] Claude E. Shannon. 1948. A mathematical theory of communication. *Bell System Technical Journal* 27, 3 (1948), 379–423.
- [37] Thomas Sikora. 1997. MPEG digital video-coding standards. *IEEE Signal Processing Magazine* 14, 5 (1997), 82–100.
- [38] Renchu Song, Weiwei Sun, Baihua Zheng, and Yu Zheng. 2014. PRESS: a novel framework of trajectory compression in road networks. In *Proceedings of the VLDB Endowment*, Vol. 7. VLDB Endowment, 661–672.
- [39] Sarah C. Stienessen and Julia K. Parrish. 2013. The effect of disparate information on individual fish movements and emergent group behavior. *Behavioral Ecology* 24, 5 (2013), 1150–1160.
- [40] Subhash Suri. 1990. On some link distance problems in a simple polygon. *IEEE Transactions on Robotics and Automation* 6, 1 (1990), 108–113.
- [41] Kevin Toohey and Matt Duckham. 2015. Trajectory similarity measures. *SIGSPATIAL Special 7*, 1 (May 2015), 43–50.
- [42] Godfried T. Toussaint. 1984. A new linear algorithm for triangulating monotone polygons. *Pattern Recognition Letters* 2, 3 (1984), 155–158.
- [43] Gregory K. Wallace. 1992. The JPEG still picture compression standard. *IEEE Transactions on Consumer Electronics* 38, 1 (1992), xviii–xxxiv.
- [44] Ian H. Witten, Radford M. Neal, and John G. Cleary. 1987. Arithmetic coding for data compression. *Commun. ACM* 30, 6 (1987), 520–540.
- [45] Dongxiang Zhang, Mengting Ding, Dingyu Yang, Yi Liu, Ju Fan, and Heng Tao Shen. 2018. Trajectory simplification: an experimental study and quality analysis. In *Proceedings of the VLDB Endowment*, Vol. 11. VLDB Endowment, 934–946.
- [46] Yu Zheng. 2015. Trajectory data mining: an overview. *ACM Transactions on Intelligent Systems Technology* 6, 3 (May 2015), 29:1–29:41.
- [47] Yu Zheng, Xing Xie, Wei-Ying Ma, et al. 2010. GeoLife: A collaborative social networking service among user, location and trajectory. *IEEE Data Engineering Bulletin* 33, 2 (2010), 32–39.
- [48] Jacob Ziv and Abraham Lempel. 1978. Compression of individual sequences via variable-rate coding. *IEEE transactions on Information Theory* 24, 5 (1978), 530–536.