High Dimensional Trajectory Simplification with Multidimensional Interpolation (Technical Report)

ANONYMOUS AUTHORS

High-dimensional trajectory data is ubiquitous due to the popularity of recommendation systems, social networks, traffic prediction, etc. It consists of a sequence of multi-dimensional points, each of which contains several features of a user at a certain time instance. The multi-dimensional point captures the user behavior, such as locations and actions, over time. Despite its popularity, the massive data size prevents its usage in many real-time applications. One effective method for mitigating the problem is trajectory simplification which compresses the trajectory data without incurring significant information loss. Most existing algorithms of trajectory simplification only focus on the two-dimensional trajectory data (which lies in 2D Euclidean space) and their core components involve the 2D geometry computation, which renders them unable to apply to high-dimensional trajectory data. The only existing high-dimensional algorithm adopts the handcrafted rules for the trajectory compression, which renders it suffering from suboptimal performance and limited application scenarios (i.e., limited error measurements). Motivated by this, this paper proposed and developed an effective algorithm, namely *high-dimensional trajectory simplification with multidimensional interpolation* (*HITS*). In particular, we decompose a d-dimensional trajectory into a set of one-dimensional trajectories, where $d \geq 2$. For each one-dimensional trajectory, we develop a reinforcement learning (RL) paradigm for this one-dimensional trajectory simplification, which enjoys a better compression ratio and general error measures. After that, we combine all the simplified one-dimensional trajectories and convert them into a d-dimensional trajectory through an interpolation algorithm. Our empirical study shows that our proposed algorithm is comparable with the best 2D trajectory simplification algorithm for two-dimensional trajectory simplification. Besides, our proposed algorithm significantly outperforms the baselines for high-dimensional trajectory simplification in terms of compression ratio and simplification error.

Additional Key Words and Phrases: Trajectory, Data Compression, Reinforcement Learning, Urban Computing

ACM Reference Format:
Anonymous Authors. 2025. High Dimensional Trajectory Simplification with Multidimensional Interpolation (Technical Report). 1, 1 (April 2025), 25 pages. https://doi.org/XXXXXXX.XXXXXXX

1 introduction

With the widespread adoption of geo-positioning technologies, trajectory data has become ubiquitous and increasingly popular, finding numerous applications in smart city and urban computing, such as route planning, ride sharing, and vehicle dispatching in car-hailing. Trajectory data captures the mobility and traces of moving objects, such as vehicles, pedestrians, and robots. Each trajectory consists of a sequence of points, with each point denoting a geo-location at a timestamp. Figure 1 shows an example of trajectory data. It consists of a sequence of points with timestamps. This data is generated and collected continuously by remote sensors

Author's Contact Information: Anonymous Authors.

Manuscript submitted to ACM

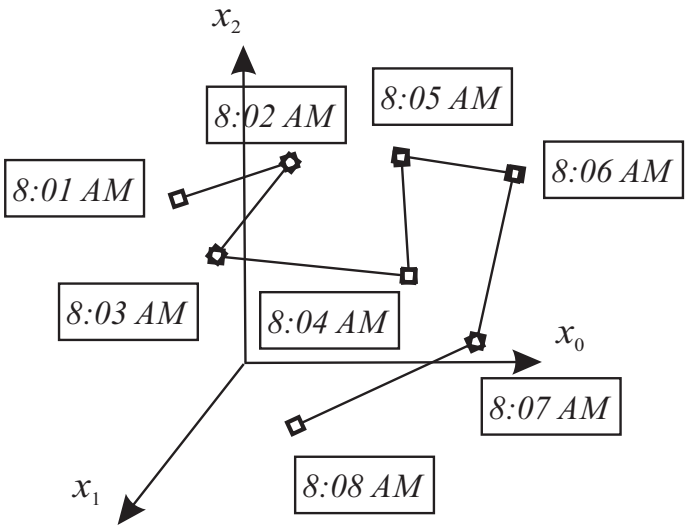

Fig. 1. An Example of High-Dim Trajectory Data

equipped on the moving object, such as GPS devices, periodically with a certain frequency (e.g., every 15 seconds). In a city such as Beijing, there are typically millions, or even billions, of moving objects (including both vehicles and pedestrians) equipped with GPS devices, generating trajectory data simultaneously. Each object generates trajectory points at a high frequency (e.g., every 15 seconds) to ensure that every location visited by the moving object is captured by the collected trajectory. Once all trajectory data is collected and accumulated into the server, the volume of data is significantly huge (see the survey [43] for more detailed illustration), resulting in a forbiddingly large space overhead, which makes subsequent data processing in applications such as trajectory similarity search, trajectory-based traffic prediction, etc. highly expensive.

Motivated by this, a considerable amount of research effort has been devoted to *trajectory simplification*, which essentially finds a compressed (i.e., simplified) version of the original trajectory [1, 2, 4–7, 10–12, 14, 16–21, 23, 25, 26, 29]. The compressed version has a much smaller number of points while having negligible information loss, which is measured by the "error" of the simplified trajectory. The study of trajectory simplification is based on the observation that raw trajectory data is often oversampled with a significantly high sampling rate, leading to a large data redundancy of collected trajectories. For instance, consider a trajectory with all points lying on a straight line or roughly on a straight line. It suffices to represent the trajectory with only two points - one being the starting point of the trajectory and the other being the endpoint of the trajectory - as any other points in the original trajectory are redundant.

Most existing works are devoted to the simplification of two dimensional (2D) trajectory, where each point is located on the 2D x-y plane. At early stage of 2D trajectory simplification research, researchers focused on developing efficient *rule-based* algorithms [1, 2, 4, 5, 10–12, 14, 16–21, 23, 25, 26, 29] that use a set of pre-defined *hand-crafted* algorithmic rules to decide whether each point in the original trajectory should be dropped or not. However, each rule-based algorithm only works for a specific error function, as there is no universal rule for all error functions. Later, reinforcement learning-based algorithms [6, 36, 37, 39] were proposed to mitigate the deficiencies of rule-based methods. They formulate trajectory simplification as a *sequential decision process*,

Table 1. Air Quality

Timestamp	$PM2.5$ (g/m^3)	$PM10$ (g/m^3)	CO (mg/m^3)	NO_2 (g/m^3)
00:00:00	24.9	77.1	2.3	25.9
01:00:00	26.2	73.4	2.6	26.1
02:00:00	27.8	78.2	2.6	27.4
03:00:00	24.3	78.3	2.8	22.1
04:00:00	21.5	78.4	2.7	20.8
05:00:00	20.4	71.9	2.6	25.7
06:00:00	24.9	67.0	2.3	25.9
07:00:00	26.2	63.4	2.2	26.1

considering all points in a trajectory one by one in chronological order. RL-based algorithms can be applied to any error function through the design of state and reward in the RL agent. However, their design for the states, rewards, and transitions (which involve complicated 2D geometry) is limited to the 2D trajectories. In practice, a plethora of real-world applications involve high-dimensional trajectory data (each point in which is a d-dimensional point, where $d > 2$). Below are some examples.

(1) **3D Vehicular Moving Trajectories.** In the urban computing and smart city applications such as ride sharing in Didi and Uber, the location information of a moving vehicle not only contains the longitudes and latitudes of the object in each timestamp but also the altitude information in each timestamp. Two typical datasets of the 3D vehicular moving trajectories are *Mopsi* [22] and *GeoLife* [45].

(2) **Air Quality.** Environmental scientists are concerned about the air quality. They deploy sensors to continuously record multiple air components, such as $PM2.5$, $PM10$, CO, NO_2, etc., in each timestamp. Table 1 shows an example. These records form a high-dimensional trajectory data, where each dimension of the point corresponds to the concentration of a component [28, 40].

(3) **Road Traffic Flow.** Road traffic is monitored using surveillance systems to gather insights for urban planning. At a busy intersection, tracking the flow of different vehicle types, e.g., cars, buses, trucks, etc., can provide valuable information. This information is recorded over time, forming a high-dimensional data, where each dimension represents the traffic flow of a specific vehicle type [42, 47].

As far as we are concerned, there is only one existing work on the high-dimensional trajectory simplification [7]. This work proposed *Synchronized L-∞ Euclidean Distance* for measuring the simplification error of a simplified high-dimensional trajectory w.r.t. the original one. Based on this distance, they developed a rule-based method for high-dimensional trajectory simplification. However, due to the nature of the rule-based method, their algorithm is limited to this Synchronized L-∞ Distance and can not be applied to general error functions. In addition, the proposed *Synchronized L-∞ Euclidean Distance* has not been employed in any other works and lacks the intuition and motivation for practical usage.

In this paper, we propose to study *high-dimensional trajectory simplification*, where each point of the simplified trajectory is a point in a multi-dimensional Euclidean space. In the high-dimensional trajectory simplification problem, we aim to find a simplified version of the given high-dimensional trajectory with the minimum simplification error. The high-dimensional trajectory simplification is a challenging problem and any existing 2D trajectory simplification algorithm can not apply to this problem since their 2D geometry component is limited

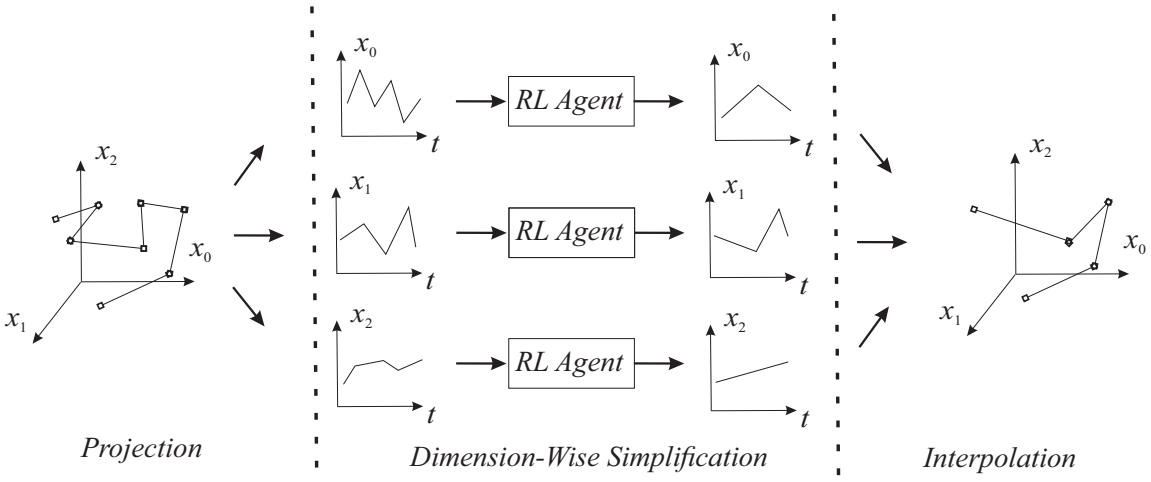

Fig. 2. Our Algorithm Framework for High Dimensional Trajectory Simplification

to the 2D trajectory data. To this end, we propose an effective algorithm, namely *high-dimensional trajectory simplification with multidimensional interpolation* (*HITS*), which employs *projection-interpolation* framework for the high-dimensional trajectory simplification. In this framework, the given d-dimensional trajectory is first projected into d one-dimensional trajectories, each of which corresponds to one dimension of the Euclidean space. Then, we adopt an *Reinforcement Learning* (*RL*) algorithm to simplify each projected one-dimensional trajectory. Finally, we propose a interpolation algorithm which integrates the simplified one-dimensional trajectory into a d-dimensional trajectory.

Our contributions are summarized as follows. First of all, we proposes a *projection-interpolation* framework for the *high-dimensional trajectory simplification*. This framework first projects the original d-dimensional trajectory into d one-dimension (each of which corresponds to one dimension of the Euclidean space,) and then, it simplifies each one-dimensional trajectory by an reinforcement learning agent. Finally, it converts all simplified one-dimensional trajectory into a d-dimensional trajectory through an interpolation algorithm. Secondly, we proposed an interpolation algorithm which combines all simplified one-dimensional trajectories into a d-dimensional trajectory. We also theoretically analyze the relationship between the error of each one-dimensional simplified trajectory and the final interpolated d-dimensional trajectory and further derived the simplification error of the final simplified d-dimensional trajectory. In addition, we also analyze the relationship between the size of each one-dimensional simplified and the final interpolated d-dimensional trajectory. The theoretical analysis provides the guidance of the parameter setting for each one-dimensional trajectory simplification. Finally, we conducted experiments on real datasets and demonstrates that our method achieves the state-of-the-art performances on the trajectory simplification.

The remainder of the paper is organized as follows. Section 2 formally presents the problem statement. Section 3 presents our proposed algorithm. Section 4 demonstrates our experiment. Section 5 presents the existing studies on the trajectory simplification and finally, Section 6 concludes this paper and proposes several research directions in the future.

2 Problem Statement

This section presents the problem statement. First, we introduce the trajectory with relevant notations. Then, the loss of trajectory simplification is discussed. Finally, we show two versions of the formal problem definition.

Notation	Physical Meaning
T	A d-dimensional raw trajectory in the format of $< T(t_1), T(t_2), \ldots, T(t_i), \ldots, T(t_n) >$.
t_i	A timestamp with the index i in the raw trajectory T.
$T(t_i)$	A d-dimensional point at timestamp t_i in trajectory T (i.e., $T(t_i) = (T(t_i)[1], T(t_i)[2], \ldots, T(t_i)[j], \ldots, T(t_i)[d])$).
$T(t_i)[j]$	The j-th coordinator value of point $T(t_i)$.
t	A non-negative real-valued number denoting continuous timestamp.
$T(t)$	A point at timestamp t in trajectory T (i.e., $T(t) = T(t_i) + \frac{t-t_i}{t_{i+1}-t_i}(T(t_{i+1}) - T(t_i))$, where $t \in [t_i, t_{i+1}]$).
$T(t)[j]$	The j-th coordinator value of point $T(t)$.
T_j	The projection of the raw trajectory T on the j-th dimension (i.e., $T_j = (T(t_1)[j], T(t_2)[j]), \ldots, T(t_i), \ldots, T(t_n)[j])$).
$T_j(t_i)$	The one-dimensional point on T_j at the timestamp $t_i, i \in [1, n]$ (i.e., $T_j(t_i) = T(t_i)[j]$).
$T_j(t_l : t_r)$	A subsequence of T_j consisting of the points $T_j(t_l), T_j(t_{l+1}), T_j(t_{l+2}), \ldots, T_j(t_r)$, where $1 \le l \le r \le n$.
$V(t)$	The velocity at timestamp t lying on one line segment in the raw trajectory T.
T'	A simplified trajectory of T_j in the form of $T' =< T'(t_{s_1}), T'(t_{s_2}), ..., T'(t_{s_k}), \ldots, T'(t_{s_m}) >$, where $1 \le s_1 \le s_2 \cdots \le s_k \cdots \le s_m \le n$.
$T'(t)$	The point at timestamp t lying on one line segment in the simplified trajectory T'.
$T'(t)[j]$	The j-th coordinator value of the point $T'(t)$.
T'_j	The projection of T_j on the j-th dimension $T'_j =< T'(t_{s_1})[j], T'(t_{s_2})[j], \ldots, T'(t_{s_k}[j]), \ldots, T'(t_{s_m})[j] >$, where $1 \le s_1 \le s_2 \cdots \le s_k \cdots \le s_m \le n$.
$\epsilon_{SED}(T'\|T)$	Synchronized Euclidean Distance (SED) between a raw trajectory T and its simplified trajectory T'.
$\epsilon_{SAD}(T'\|T)$	Speed-Aware Distance (SAD) between a raw trajectory T and its simplified trajectory T'.

Table 2. Concepts and Notations

A trajectory, denoted by T, is the path of a moving object in space as a function of time. Formally, it is a function $T(t)$: $\mathbb{R}_+ \to \mathbb{R}_+^d$, where t denotes a timestamp and $T(t) = (T(t)[1], T(t)[2], ..., T(t)[j], ..., T(t)[d])$ is a d-dimensional point that represents the location of the object in space at timestamp t. Each $T(t)[j]$ denotes the j-th coordinator value of the location. In the following, we use "location/point" interchangeably.

In practice, it is difficult to explicitly show function $T(t)$ by a formula since some trajectories might be complex. Thus, we describe a trajectory with a few sampled points $< T(t_1), T(t_2), ..., T(t_i), ..., T(t_n) >$, where t_i are timestamps and $t_i < t_{i+1}$. Any other point in the trajectory can be represented by the sampled points through linear interpolation [26]. Specifically, consider a timestamp $t \in [t_i, t_{i+1}]$. Let us use $\overline{T(t_i)T(t_{i+1})}$ to represent the line segment from point $T(t_i)$ to point $T(t_{i+1})$. We assume that the object moves at a constant *velocity* on line segment $\overline{T(t_i)T(t_{i+1})}$. Thus, for point $T(t)$, we have

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

and the the *velocity* $V(t)$ of the object at timestamp t is

$$V(t) = \frac{T(t_{i+1}) - T(t_i)}{t_{i+1} - t_i} \tag{2}$$

Definition 1 (Trajectory). *A trajectory, denoted by T, is a sequence of points $< T(t_1), T(t_2), ..., T(t_i), ..., T(t_n) >$, where t_i are timestamps ($t_i < t_{i+1}$). For any timestamp $t \in [t_i, t_{i+1}]$, point $T(t)$ and velocity $V(t)$ can be calculated by Equations 1 and 2, respectively.*

The cardinality of T, denoted by $|T|$, is the number of points in T, i.e., $|T| = n$. The simplified trajectory of T has a smaller cardinality than T.

Definition 2 (Simplified Trajectory). *Given a trajectory T, a simplified trajectory of T, denoted by T', is a trajectory that approximates T with fewer points, i.e., $T' =< T'(t_{s_1}), T'(t_{s_2}), ..., T'(t_{s_k}), ..., T'(t_{s_m}) >$, where $1 \le s_1 \le s_2 ... \le s_k ... \le s_m \le n$.*

Consider a point $T'(t_{s_k})$ with timestamp t_{s_k} in the simplified trajectory T', where $k \in [1, m]$. We call the point $T(t_{s_k})$ with the timestamp t_{s_k} in the raw trajectory T *the corresponding point* of $T'(t_{s_k})$ in T. Similarly, $T'(t_{s_k})$ is also called *the corresponding point* of $T(t_{s_k})$. Note that $T'(t_{s_k})$ may or may not be the same as $T(t_{s_k})$.

In the simplified trajectory T', any point can also be represented by the sampled points through linear interpolation. Consider a non-negative real-valued timestamp $t \in [t_{s_1}, t_{s_m}]$. Let t_{s_k} denote the timestamp, where $t_{s_k} \le t \le t_{s_{k+1}}$. The point at time t in the simplified trajectory T' is defined as follows.

$$T'(t) = T'(t_{s_k}) + \frac{t - t_{s_k}}{t_{s_{k+1}} - t_{s_k}} (T'(t_{s_{k+1}}) - T'(t_{s_k})) \tag{3}$$

Given a raw trajectory T and one simplified trajectory T' of T, we denote the loss of T' compared to T by $\epsilon(T'|T)$. There are many different error functions proposed to measure the loss. In this paper, we consider the most fundamental and commonly applied error functions for trajectory simplification, namely *Synchronized Euclidean Distance* (*SED*) [23, 25, 27, 29] and *Speed-Aware Distance* (*SAD*) [26]. Their major ideas are similar. Consider a point $T(t)$ with timestamp t in T. The error functions first find the corresponding points $T'(t)$ with the same timestamp t in T'. Then, they calculate the loss between line segments $T(t)$ and $T'(t)$. Each of them has its specific calculation method. The total loss is the maximum loss across all points $T(t)$. Their definitions are shown as follows.

SED. *The Synchronized Euclidean Distance* (*SED*) measures the maximum *point difference* between a raw trajectory T and its simplified trajectory T'. Consider points $T(t)$ and $T'(t)$. The loss between these two points is defined as follows.

$$\begin{aligned} \epsilon_{SED}(T'(t)|T(t)) &= ||T(t) - T'(t)||_2 \\ &= \sqrt{\textstyle\sum_{j=1}^{d} (T(t)[j] - T'(t)[j])^2} \end{aligned} \tag{4}$$

Consider any adjacent pair of timestamps t_i and t_{i+1} in T. Since $\overline{T(t_i)T(t_{i+1})}$ and $\overline{T'(t_i)T'(t_{i+1})}$ are line segments, the maximum point difference must achieve at the endpoints of line segments. Thus, we only need to consider timestamps t_i and t_{i+1}, rather than all timestamps $t \in [t_i, t_{i+1}]$.

Then, the definition of SED can be formalized.

Definition 3 (Synchronized Euclidean Distance (SED)). *Given a raw trajectory T and its simplified trajectory T', the SED error $\epsilon_{SED}(T'|T)$ between T and T' is*

$$\begin{aligned}\epsilon_{SED}(T'|T) &= \max_{t\in\{t_1,t_2,\ldots,t_n\}} ||T(t) - T'(t)||_2 \\ &= \max_{t\in\{t_1,t_2,\ldots,t_n\}} \sqrt{\sum_{j=1}^{d}(T(t)[j] - T'(t)[j])^2}\end{aligned} \tag{5}$$

SAD. *The Speed-Aware Distance (SAD)* measures the maximum *velocity difference* between a raw trajectory T and its simplified trajectory T'. It extends the concept of velocity difference by incorporating orthogonal decomposition in a multi-dimensional coordinate system. CThe loss between these two points is defined as follows.

$$\begin{aligned}\epsilon_{SAD}(T'(t)|T(t)) &= ||V(t) - V'(t)||_2 \\ &= \sqrt{\sum_{j=1}^{d}(V(t)[j] - V'(t)[j])^2}\end{aligned} \tag{6}$$

Consider any adjacent pair of timestamps t_i and t_{i+1} in T. Based on Equation 2, the velocity $V(t)$ (resp. $V'(t)$) along line segments $\overline{T(t_i)T(t_{i+1})}$ (resp. $\overline{T'(t_i)T'(t_{i+1})}$) are the same. Thus, we only need to consider timestamps t_i or t_{i+1}, rather than all timestamps $t \in [t_i, t_{i+1}]$. The definition of SAD is shown below.

Definition 4 (Speed-Aware Distance (SAD)). *Given a raw trajectory T and its simplified trajectory T', the SAD error $\epsilon_{SAD}(T'|T)$ between T and T' is*

$$\begin{aligned}\epsilon_{SAD}(T'|T) &= \max_{t\in\{t_1,t_2,\ldots,t_n\}} ||V(t) - V'(t)||_2 \\ &= \max_{t\in\{t_1,t_2,\ldots,t_n\}} \sqrt{\sum_{j=1}^{d}(V(t)[j] - V'(t)[j])^2}\end{aligned} \tag{7}$$

The trajectory simplification problem has two variants, namely *Min-Error (ME) problem* and *Error-Bounded (EB) problem*. Note that in the two definitions below, the error function $\epsilon(T'|T)$ can be either *Synchronized Euclidean Distance (SED)* $\epsilon_{SED}(T'|T)$ or *Speed-Aware Distance (SAD)* $\epsilon_{SAD}(T'|T)$.

Problem 1 (Min-Error (ME)). *Given a raw trajectory T,* Min-Error problem *finds the simplified trajectory T' of T such that the error $\epsilon(T'|T)$ of T' is minimized and the cardinality $|T'|$ is within a given budget θ.*

Problem 2 (Error-Bounded (EB)). *Given a raw trajectory T,* Error-Bounded problem *finds the simplified trajectory T' of T with the minimum number of sampled points such that the error $\epsilon(T'|T)$ is within a given threshold δ.*

3 Our Proposed Method

This section presents our proposed algorithm, namely *high-dimensional trajectory simplification with multidimensional interpolation (HITS)*. It has two variants called *HITS-ME* and *HITS-EB* and they are designed to tackle the Min-Error (ME) problem and Error-Bounded (EB) problem, respectively. Both *HITS-ME* and *HITS-EB* follow the same algorithm framework which we demonstrate in Section 3.1. Section 3.2 presents our interpolation methods which combines all simplified one-dimensional trajectories. Then, Section 3.3 presents our algorithm for one-dimensional trajectory simplification. Finally, Section 3.4 presents the theoretical analysis on the simplification error, the compression ratio and the time complexities of our proposed algorithms.

3.1 Algorithm Framework

Figure 2 demonstrates the framework of our algorithm with a concrete example. The leftmost sub-figure shows the original trajectory with dimensionality $d = 3$. The three dimensions/axis in the example is denoted by x_0, x_1 and x_2, respectively. In the middle sub-figure, we decompose the original trajectory into three separate trajectories, each corresponding to one of the dimensions x_0, x_1, and x_2. As a result, there are three one-dimensional trajectories. We formulate each one-dimensional trajectory simplification as a *Markov Decision Process* (*MDP*) and tackle it with the *reinforcement learning* (*RL*) algorithm. In particular, we train an RL agent for one-dimensional trajectory simplification and simplify each one-dimensional trajectory with the trained RL agent. In the rightmost sub-figure, we propose an interpolation algorithm that combines all simplified one-dimensional trajectories into a d-dimensional trajectory.

We also theoretically prove the correspondence between the simplification error of one-dimensional simplified trajectories and the final interpolated d-dimensional trajectory. As such, the error of the final result can be guaranteed if the error parameter in the one-dimensional RL agent is set properly.

Our overall algorithms for Min-Error and Error-bounded problems of the high-dimensional trajectory simplification are shown as follows.

3.1.1 Min-Error High-Dimensional Trajectory Simplification. Algorithm 1 shows the pseudocode of our algorithm called *HITS-ME* for the Min-Error high-dimensional trajectory simplification problem. Given a d-dimensional original trajectory T and a budget θ (which is a positive integer), we first decompose T into d one-dimensional trajectories $(T_1, T_2, ..., T_d)$ and for each one-dimensional trajectory T_j, $j \in [1, d]$, we invoke the one-dimensional simplification algorithm 1D-ME (to be presented later) to obtain a simplified one-dimensional trajectory T'_j. It is worth noting that to ensure that the final d-dimensional simplified trajectory has at most W points, we set the budget W in each 1D-ME to be $\frac{\theta}{d}$. Finally, we invoke the Interpolation algorithm (Algorithm 3) to obtain the final d-dimensional simplified trajectory T' from the d simplified one-dimensional trajectories $T'_1, T'_2, ..., T'_d$.

3.1.2 Error-Bounded High-Dimensional Trajectory Simplification. Algorithm 2 shows the pseudocode of our algorithm called *HITS-EB* for the Error-Bounded high-dimensional trajectory simplification problem. Given a d-dimensional original trajectory T and an error parameter ϵ, we first decompose T into d one-dimensional trajectories $(T_1, T_2, ..., T_d)$ and for each one-dimensional trajectory T_j, $j \in [1, d]$, we invoke the one-dimensional simplification algorithm 1D-ME (to be presented later) to obtain a simplified one-dimensional trajectory T'_j. It is worth noting that to ensure the simplification error in the final d-dimensional simplified trajectory, we set the error parameter ϵ in each 1D-ME algorithm to be $\frac{\delta}{\sqrt{d}}$. Finally, we invoke the Interpolation algorithm (Algorithm 3) to obtain the final d-dimensional simplified trajectory T' from the d simplified one-dimensional trajectories $T'_1, T'_2, ..., T'_d$.

3.2 High-Dimensional Simplification via Interpolation

Algorithm 3 demonstrates our interpolation algorithm. Given d one-dimensional trajectories $T'_1, T'_2, ..., T'_d$, it outputs a d-dimensional trajectory T'. We will demonstrate in the theoretical analysis that the projection of T' in the j-th dimension coincides with T'_j. We initialize T' to be T'_1 in Line 1. Then, in the for-loop in Lines 2-23, we will iteratively consider other dimensions one by one. Consider the j-th iteration in the for-loop. We first

Algorithm 1: HITS-ME(T, θ)

Require: The original d-dimensional trajectory T, the budget θ
Ensure: A d-dimensional trajectory T'
// One-Dimensional Trajectory Simplification via Deep Reinforcement Learning
1: **for** j = 1, 2, 3, ..., d **do**
2: $T'_j \leftarrow$ 1D-ME(T_j, $\frac{\theta}{d}$);
3: **end for**
// Interpolation
$T' \leftarrow$ Interpolation($T'_1, T'_2, T'_3, ..., T'_d$);
4: Return T';

Algorithm 2: HITS-EB(T, δ)

Require: The original d-dimensional trajectory T, the error parameter δ
Ensure: A d-dimensional trajectory T'
// One-Dimensional Trajectory Simplification via Deep Reinforcement Learning
1: **for** j = 1, 2, 3, ..., d **do**
2: $T'_j \leftarrow$ 1D-EB(T_j, $\frac{\delta}{\sqrt{d}}$);
3: **end for**
// Interpolation
$T' \leftarrow$ Interpolation($T'_1, T'_2, T'_3, ..., T'_d$);
4: Return T';

initialize a j-dimensional trajectory $\mathcal{T}$ to be $\emptyset$. Consider the for-loop in Lines 4-20. In each iteration, we consider a point $T'[t_k]$ at the timestamp t_k in T'. In Lines 5-7, we simply copy the j'-th coordinate value of $T'(t_k)$ to $\mathcal{T}(t_k)$ for each $j' \in [1, j-1]$. In Lines 9-10, we will calculate the j-th coordinate value of $\mathcal{T}'(t_k)$ and we adopt a linear interpolation between $T'_j(t_\tau)[j]$ and $T'_j(t_\tau + 1)[j]$. In Lines 11-19, we consider each point p in T'_j whose timestamp is between t_k and t_{k+1}. Let t denote the timestamp of p. For the first $j-1$ coordinate values of $\mathcal{T}'(t)$, we adopt a linear interpolation between $T'(t_k)$ and $T'(t_{k+1})$. The j-th coordinator value of $\mathcal{T}'(t)$ is assigned to be $T'_j(t)$. In Lines 21-22, we sort all points in $\mathcal{T}$ in the chronological order and assign T' to be $\mathcal{T}$. After that, we go to the next iteration and consider the next dimension. Finally, Line 24 returns the d-dimensional simplified trajectory T'.

3.3 One-Dimensional Compression via Reinforcement Learning

In this section, we present our *reinforcement learning* (*RL*) *paradigm* for each one-dimensional trajectory compression. We first briefly introduce the preliminaries of *reinforcement learning* and then, we model each version of the trajectory compression as a *Markov decision process* (*MDP*) first.

3.3.1 Preliminaries of Reinforcement Learning. Reinforcement learning (RL) was initially introduced to guide agents on the optimal actions to be taken in a particular environment, aimed at maximizing the cumulative reward [32]. Typically, the environment is modeled as a *Markov decision process* (*MDP*) [30]. In recent years, RL models have been employed efficiently to address algorithmic problems.

In the reinforcement learning paradigm, there are two components, namely *agent* and *environment*, and the agent tries to obtain as large accumulative rewards as possible through the interaction with the environment. Specifically, the environment consists of *states*, *rewards*, *actions*, *transitions* and a *decay factor*. The states defines

Algorithm 3: Interpolation(T'_1, T'_2, ..., T'_d)

Require: d 1-dimensional simplified trajectories T'_1, T'_2, ..., T'_d
Ensure: A d-dimensional trajectory T'
1: $T' \leftarrow T'_1$;
2: **for** j = 2, 3, ..., d **do**
3: Initialize a j-dimensional trajectory $\mathcal{T}$ to be $\emptyset$;
4: **for** k = $s_1, s_2, ..., s_{|T'|-1}$ **do**
5: **for** j'=1,2,...,j-1 **do**
6: $\mathcal{T}(t_k)[j'] \leftarrow T'(t_k)[j']$;
7: **end for**
8: Find the point $T'_j(t_\tau)$ such that $t_\tau \leq t_k < t_{\tau+1}$;
9: $\omega \leftarrow \frac{T'_j(t_{\tau+1}) - T'_j(t_\tau)}{t_{\tau+1} - t_\tau} \cdot (t_k - t_\tau)$;
10: $\mathcal{T}[j](t_k) \leftarrow \omega$;
11: Find the set $\mathcal{S}$ of points in T'_j whose timestamps are between t_k and t_{k+1};
12: **for** Each point p in $\mathcal{S}$ **do**
13: $t \leftarrow$ the timestamp of p;
14: Create a new point $p' \leftarrow \frac{T'(t_{k+1}) - T'(t_k)}{t_{k+1} - t_k} \cdot (t - t_k)$;
15: **for** j'=1,2,...,j-1 **do**
16: $\mathcal{T}(t)[j'] \leftarrow p'[j']$;
17: **end for**
18: $\mathcal{T}(t)[j] \leftarrow T'_j(t)$;
19: **end for**
20: **end for**
21: Sort the points in $\mathcal{T}$ in the chronological order;
22: $T' \leftarrow \mathcal{T}$;
23: **end for**
24: Return T';

all possible statuses that the agent stays, the actions provides the choices that the agent could take in each state. Transitions defines how the next state that the agent transits to after an action is taken. For each pair of state and action, there is a reward r given to the agent and the intermediate reward obtained is calculated as $r \times d^t$, where d denotes the decay factor and t denote the number of the steps that the agent takes from the beginning. Finally, the accumulative reward is the sum of all intermediate rewards obtained in all steps. A policy is defined as a mapping from each state to an action. It is worth mentioning that given a fixed policy, the environment is a *Markov decision process* (*MDP*). From the description above, we could observe that the task of the agent is to learn an optimal policy which provides the optimal action given the current state such that the accumulative rewards could be maximized.

3.3.2 Min-Error Problem. Given a storage budget W (which is a positive integer), the *Min-Error* problem aims to find a simplified trajectory T'_j of the raw trajectory T_j, where $j \in [1, d]$, such that $|T'_j| \leq W$ and $\epsilon(T'_j|T_j)$ is minimized. The error function $\epsilon(\cdot|\cdot)$ considered here can be but not limited to SED and SAD. It is worth mentioning that in the RL algorithm for Min-Error problem, we consider all points in T_j in the chronological order. We maintain a buffer $\mathcal{B}^i$ with the size equal to W. Initially, we put the first W points into $\mathcal{B}^i$ and we linear scan all remaining points in the chronological order. For each point currently considered (denoted by $T_j(t_i)$), we assign $\mathcal{B}^{i+1}$ to be $\mathcal{B}^i$ and compute the state of $T_j(t_i)$ and the states of each point in the buffer $\mathcal{B}^i$. Based on these states and the policy learned, we will sample an action which is to drop one point in the buffer

$\mathcal{B}^i$ from the RL policy network $\pi_\Theta(a|S^i)$ which is represented by a neural network parameterized with Θ (to be presented later). After that, we insert the currently point $T_j(t_i)$ into $\mathcal{B}^i$. Note that when $i+1=n$, it inserts the last point ($T_j[n]$) into the buffer and thus, our algorithm ensures that the last point must be included in T'_j. Then, we increase i by 1 and go to the next point. Finally, we will return the list of all points in the buffer $\mathcal{B}^n$ as the simplified one-dimensional trajectory T'_j after we scan and consider the last point $T_j[n]$. This ensures that the size of the T'_j is at most W. The detailed pseudocode is shown in Algorithm 4 and the states, actions, transitions and reward of our RL agent are defined as follows.

- *States*: Consider the i-th point $T_j(t_i)$ in the one-dimensional raw trajectory T_j, where there are n points in T_j. Our definition for the state of the point is inspired by the heuristic introduced in [36]. We define the *error value* for each point $q^i_\phi (1 \le \phi \le W)$ in $\mathcal{B}^i$, denoted by $e_{q^i_\phi}$, as the maximum error between the original trajectory T_j and the trajectory comprised of the first ϕ points in the buffer $\mathcal{B}^i$. In particular, let $\mathcal{B}^i_\phi$ denote the list of the first ϕ points in $\mathcal{B}^i$ whose timestamp is smaller or equal to the timestamp of q^i_ϕ. Without loss of generality, we assume that all points in $\mathcal{B}^i_\phi$ are sorted in chronological order. Then, the formula of $e_{q^i_\phi}$ is shown as follows.

$$e_{q^i_\phi} = \epsilon(\mathcal{B}^i_\phi|T) \tag{8}$$

  The *State* of the i-th point $T_j(t_i)$ is:

$$S^i = e_{q^i_1}, e_{q^i_2}, \ldots, e_{q^i_W} \tag{9}$$

- *Actions*: The action space contains W actions, corresponding to choosing $\mathcal{B}^i[\phi](1 < \phi \le W)$ to drop. An action, denoted by a_i, is defined as:

$$a_i := \phi^*_i \quad (1 < \phi^*_i \le W) \tag{10}$$

  where action $a_i = \phi^*_i$ means choosing the ϕ^*_i-th point in $\mathcal{B}^i$ to drop. According to our action, the first point in the buffer can not be dropped and we always keep the first point in $\mathcal{B}^i$. Note that in the first iteration in the for-loop in Lines 6-14 (when $i = W+1$), the first point in the buffer is $T_j[1]$ and as as result, our algorithm always includes the point $T_j[1]$ in the final simplified one-dimensional trajectory T'_j.
- *Transitions*: When an action a_i is taken, the point to be dropped from the buffer $\mathcal{B}^i$ is determined. The new state, denoted by S^{i+1}, can be computed as follows. Once the point to be dropped is decided, the error value of each point is updated based on the maximum error between the original and simplified trajectory in the current buffer $\mathcal{B}^i$ and the error value of the chosen point $q^i_{a_i}$. Here, the error value of the new state S^{i+1} is: $e_{q^{i+1}_\phi} = \max(e_{q^i_{a_i}}, e_{a_i, q^{i+1}_\phi})$ if $\phi > a_i$ and $e_{q^{i+1}_\phi} = e_{q^i_\phi}$ otherwise, where e_{a_i, q^{i+1}_ϕ} denotes the error between the original trajectory T and the subsequence $\mathcal{B}^i[a_i : \phi]$ of $\mathcal{B}^i$.
  Thus, the new state S^{i+1} is:

$$S^{i+1} = e_{q^{i+1}_1}, e_{q^{i+1}_2}, \ldots, e_{q^{i+1}_W} \tag{11}$$

- *Rewards*: Consider taking an action $a_i = \phi^*_i$ at state S^i and arriving at state S^{i+1}. We define the reward associated with the transition from state S^i to state S^{i+1}, denoted by r^i, as:

$$r^i = -(e_{q^{i+1}_W} - e_{q^i_W}) \tag{12}$$

The intuition is that a smaller increase in the error at the i-th iteration leads to a larger reward. This definition guides the model to choose end points that minimize the growth of the error in the simplified trajectory. The goal of minimizing the overall error is thus satisfied. Suppose the agent goes through a sequence of states, each of which receives a reward. The total reward is:

$$\sum_{i=W}^{n-1} r^i = \sum_{i=W}^{n-1} -(e_{q_W^{i+1}} - e_{q_W^i}) = -e_{q_W^n} + e_{q_W^W} \tag{13}$$

Note that $e_{q_W^n}$ is equal to $\epsilon(T'_j|T_j)$ by definition and $e_{q_W^W}$ can be considered as a constant since it is always the same no matter what policy of the RL agent is. Thus, the objective of the RL agent is to minimize $\epsilon(T'_j|T_j)$.

The main problem of the MDP above is to find a *policy* for the agent. The policy is equivalent to a function that decides the actions for the agent to take under a certain *state* to maximum goal reward. We apply the *policy gradient* (PNet) method, implemented through a neural network known as PNet, to learn the policy. PNet encodes the probability $\pi_\Theta(a|s)$ using its network parameters, where $\pi_\Theta(a|s)$ represents the probability of selecting action a given the state s, where θ denotes the set of parameters of this model.

$$\pi_\Theta(a|s) = \sigma(\mathbf{A}s + b) \tag{14}$$

Here, σ denotes a softmax function, and $\Theta = (\mathbf{A}, b)$ denotes the parameters of the neural network. Then, the PNet transfers the original gradient to the gradient of parameter θ:

$$\nabla_\Theta J(\Theta) = \sum_{t=1}^{N} \frac{R_t - \bar{R}}{\sigma_R} \nabla \ln \pi_\Theta(a_t|s_t) \tag{15}$$

In this equation, $s_1, s_2, \ldots, s_N$ denotes a sequence of states explored by the agent, and $a_1, a_2, \ldots, a_N$ denotes a sequence of actions sampled by probability $\pi_\Theta(a_t|s_t)$. And R_t is the accumulated rewards, $\bar{R}$ is the mean of all R_t and σ_R is the standard deviation of R_t. To maximum the goal reward, PNet uses gradient ascent to upgrade parameters repeatedly until satisfying some criterion to stop.

3.3.3 Error-bound Problem. Firstly, given an error tolerance ϵ and a one-dimensional trajectory T_j, where $j \in [1, d]$, the Error-Bounded problem aims to find a simplified trajectory T'_j of the raw trajectory T_j with the minimum size such that the simplification error $\epsilon(T'_j|T_j)$ is within the threshold ϵ. The same as the Min-Error problem, the simplification error function $\epsilon(\cdot|\cdot)$ can be but not limited to *Synchronized Euclidean Distance* (*SED*) or *Speed-Aware Distance* (*SAD*). Therefore, we model the problem as an MDP, and define states, actions, transitions and rewards respectively. We denote by $T_j(t_l : t_r)$ a subsequence of T_j consisting of the points $T_j(t_l), T_j(t_{l+1}), T_j(t_{l+2}), \ldots, T_j(t_r)$. In this RL agent, we consider the points in the chronological order and maintain a sliding window $\mathcal{W}(= T_j[l : r])$. We also maintain a positive integer m, where $l \le m \le r$ in the linear scan of all points and we make sure that the removal of the points $T_j(t_{l+1}), T_j(t_{l+2}), \ldots, T_j(t_{m-1})$ will not violate the error constraint. That is, the error between subtrajectory $\overline{T_j(t_l)T_j(t_m)}$ (which is simply a line segment) and the subtrajectory consisting of $T_j(t_l), T_j(t_{l+1}), \ldots, T_j(t_m)$ is at most ϵ.

Algorithm 5 presents the pseudocode of the one-dimensional trajecotry simplification algorithm for Error-Bounded problem. Initially, we assign the list $\mathcal{W}$ (i.e., the sliding window of points) to be $\{T_j(t_1), T_j(t_2)\}$ and initialize l, m, r to be $1, 2, 2$, respectively. Then, we initialize the one-dimensional simplified trajectory T'_j to be $\emptyset$. Lines 4-18 presents a for-loop which slides the windows and consider the points in the chronological order. In each iteration, we first check if the error between the line segment $\overline{T_j(t_l)T_j(t_m)}$ and the subtrajectory consisting of

Algorithm 4: 1D-ME(T_j, W)

Require: A One-dimensional trajectory T_j, the budget W
Ensure: A simplified One-dimensional trajectory T'_j
1: Initialize a buffer $\mathcal{B}^0$ to be $\emptyset$;
2: **for** $i = 1, 2, ...,$ W **do**
3: $\quad \mathcal{B}^{i+1} \leftarrow \mathcal{B}^i \cup \{T_j(t_i)\}$
4: $\quad t \leftarrow t + 1$;
5: **end for**
6: **for** $i =$ W+1, W+2, ..., n **do**
7: $\quad \mathcal{B}^i \leftarrow \mathcal{B}^{i-1}$;
8: $\quad$ Compute the values of each point in $\mathcal{B}^i$;
9: $\quad$ Update the state S^i accordingly;
10: $\quad$ Sample an action $a_i \sim \pi_\Theta(a|S^i)$;
11: $\quad$ Drop the point whose timestamp is equal to a_t in $\mathcal{B}^i$;
12: $\quad$ Insert the point $T_j(i)$ into $\mathcal{B}^i$;
13: $\quad i \leftarrow i + 1$;
14: **end for**
15: $T'_j \leftarrow$ all points in $\mathcal{B}^n$;
16: Sort points in T'_j in chronological order;
17: Return T'_j;

Algorithm 5: 1D-EB(T_j, ϵ)

Require: A One-dimensional trajectory T_j, the error parameter ϵ
Ensure: A simplified One-dimensional trajectory T'_i
1: Initialize a list $\mathcal{W}$ to be $\{T_j(t_1), T_j(t_2)\}$;
2: $l \leftarrow 1, m \leftarrow l + 1, r \leftarrow 2$;
3: $T'_j \leftarrow \emptyset$;
4: **while** True **do**
5: $\quad$ **if** $\epsilon(\overline{T_j(t_l)T_j(t_r)}|T_j(t_l : t_r)) \leq \epsilon$ **then**
6: $\quad\quad$ **if** $r = n$ **then**
7: $\quad\quad\quad$ Insert $T_j(t_r)$ into T'_j;
8: $\quad\quad\quad$ Break;
9: $\quad\quad$ **end if**
10: $\quad\quad$ Update the state $S^{\mathcal{W}}$ accordingly ($\mathcal{W} = T_j(t_l : t_r)$);
11: $\quad\quad$ Sample an action $a \sim \arg\max_a Q(S^{\mathcal{W}[l:r]}, a|\Theta)$;
12: $\quad\quad \mathcal{W} \leftarrow T_j(t_l : t_r) \cup \{T_j(t_{r+1}), T_j(t_{r+2}), \ldots, T_j(t_{r+a})\}$;
13: $\quad\quad m \leftarrow r, r \leftarrow \min\{r + a, n\}$;
14: $\quad$ **else**
15: $\quad\quad$ Insert $T_j(t_m)$ into T'_j;
16: $\quad\quad l \leftarrow m$;
17: $\quad$ **end if**
18: **end while**
19: Return T'_j;

$T_j(t_l), T_j(t_{l+1}), \ldots, T_j(t_m)$ is at most ϵ. If so, we obtain that the removal of the points $T_j(t_{l+1}), T_j(t_{l+2}), \ldots, T_j(t_{m-1})$ will not violate the error constraint and go to the operations in Lines 5-12 and expand the window to see if we can drop more points without violating the error bound. If the current window can not be expanded (i.e., $r = n$), then, we insert $T_j(t_r)$ into T'_j and break the for-loop. If r is not n, we update the state $S^{\mathcal{W}}$ accordingly (to be presented later). Then, we sample an action based on the state and a neural network $Q(\cdot|\Theta)$ parameterized

by Θ for the action sampling (to be presented later). After that, we update the window $\mathcal{W}$ to be $T_j(t_l : t_{r+a})$ (i.e., the window is expanded by a points) and update m to be r and update r to be $\min\{r_a, n\}$. If the error between the line segment $\overline{T_j(t_l)T_j(t_m)}$ and the subtrajectory consisting of $T_j(t_l), T_j(t_{l+1}), \ldots, T_j(t_m)$ is less than ϵ, we insert $T'_j(t_m)$ into T'_j and update l to be m.

The only left issue is the setting of state, action, transition and reward in the RL which we detail as follows.

- *States*: Our definition for the state of the point is inspired by the heuristic introduced in [39]. Consider the window $\mathcal{W}$, the state of the current window $\mathcal{W}(= T_j(t_l : t_r))$ is defined as follows.

$$S^{\mathcal{W}} = (\frac{\sum_{h=l}^{r-1} |T_j(t_h) - T_j(t_{h+1})|}{|T_j(t_l) - T_j(t_r)|}, r - l + 1) \tag{16}$$

- *Actions*: We define an action space containing K actions, meaning to expand the window $\mathcal{W} = (T_j(t_l : t_r))$ by $a(1 \le a \le K)$, where K is a positive integer and it is a hyperparameter of our RL agent. Here, an action, denoted by a, is defined as follows:

$$a := \phi^* \quad (1 \le \phi^* \le K) \tag{17}$$

  where the action $a = \phi^*$ means it chooses to expand the window by ϕ^* points.

- *Transitions*: When an action a is taken, then we simply expand the window $\mathcal{W}$ from $(T_j(t_l : t_r))$ to the window $T_j(t_l : t_{r+a})$ (i.e., $\mathcal{W} \leftarrow T_j(t_l : t_r) \cup \{T_j(t_{r+1}), T_j(t_{r+2}), \ldots, T_j(t_{r+a})\}$). The state $S^{T_j(t_l:t_r)}$ will also be converted to state $S^{T_j(t_l:t_{r+a})}$.
- *Rewards*: Consider we take an action $a = \phi^*$ which simply expands the window $\mathcal{W}$ from $(T_j(t_l : t_r))$ to the window $T_j(t_l : t_{r+a})$. We define the reward associated with transition from state $S^{T_j(t_l:t_r)}$ to state $S^{T_j(t_l:t_{r+a})}$, denoted by r^i, as follows.

$$r_a = a - T'_j \cap T_j(t_r : t_{r+a}) \tag{18}$$

  Note that $T'_j \cap T_j(t_r : t_{r+a})$ is the number of points in the window $T_j(t_r : t_{r+a})$ which is finally inserted into T'_j. The intuition is that the number of simplified points in $T'_j \cap T_j(t_r : t_{r+a})$ is fewer, the reward is larger. So, this definition will lead the model to choose those end points that lead to fewer point in simplified trajectory. And the goal of this problem is also satisfied. Suppose the agent went through a sequence of states, each of which receives a reward. In the case that the future rewards are not discounted, we can get:

$$\sum_a r_a = \sum_a (a - T'_j \cap T_j(t_r : t_{r+a})) = n - |T'_j| \tag{19}$$

We employ the Deep-Q-Network (DQN) [24] for learning the policy of the RL agent in the MDP above. Given an action a and a state s, the DQN, denoted by $Q(s, a|\Theta)$ (also known as *Q function*), aims to represent the maximum expected accumulated reward it will receive by following any policy if the action a is taken at the state s. The Q function $Q(\cdot, \cdot|\Theta)$ is implemented by a neural network parameterized by the set Θ of trainable parameters. We refer the readers to [24] for the details.

3.4 Theoretical Analysis

Theorem 1. *Given an original trajectory T and a simplified trajectory T', recall that the* Synchronized Euclidean Distance (SED) *is defined as follows.*

$$\epsilon_{SED}(T'|T) = \max_{t\in[t_1,t_n]} \sqrt{\sum_{j=1}^{d} |T(t)[j] - T'(t)[j]|^2} \tag{20}$$

Consider the j-th dimension, we denote the SED between the projections of T and that of T' in the j-th dimension by $\epsilon_{SED}(T'|T)[j]$ (i.e., $\epsilon_{SED}(T'|T)[j] = \epsilon(T'_j|T_j)$).

$$\epsilon_{SED}(T'|T)[j] = \max_{t\in[t_1,t_n]} \sqrt{|T(t)[j] - T'(t)[j]|^2} \tag{21}$$

Then, it holds that $\epsilon_{SED}(T'|T)$ is at most δ if for each dimension j, $\epsilon_{SED}(T'|T)[j] \le \frac{\delta}{\sqrt{d}}$, where d is the total number of dimensions.

Proof. Consider the SED between a raw trajectory T and its simplified trajectory T'.

$$\begin{aligned}
&\epsilon_{SED}(T'|T)\\
=&\max_{t\in[t_1,t_n]} \sqrt{\sum_{j=1}^{d} |T(t)[j] - T'(t)[j]|^2}\\
\le&\max_{t\in[t_1,t_n]} \sqrt{d\cdot\max_{j\in[1,d]} |T(t)[j] - T'(t)[j]|^2}\\
=&\sqrt{d}\max_{t\in[t_1,t_n]} \sqrt{\max_{j\in[1,d]} |T(t)[j] - T'(t)[j]|^2}\\
=&\sqrt{d}\max_{t\in[t_1,t_n]}\max_{j\in[1,d]} \sqrt{|T(t)[j] - T'(t)[j]|^2}\\
=&\sqrt{d}\max_{j\in[1,d]}\max_{t\in[t_1,t_n]} \sqrt{|T(t)[j] - T'(t)[j]|^2}\\
=&\sqrt{d}\max_{j\in[1,d]} \epsilon_{SED}(T'|T)[j]
\end{aligned} \tag{22}$$

From Inequality (22), we obtain that if $\epsilon_{SED}(T'|T)[j] \le \frac{\delta}{\sqrt{d}}$ for any dimension j, $\epsilon_{SED}(T'|T)$ must be less than or equal to δ.

Theorem 2. *Given an original trajectory T and a simplified trajectory T', recall that the* Speed-Aware Distance (SAD) *is defined as follows.*

$$\epsilon_{SAD}(T'|T) = \max_{t\in\{t_1,t_2,\ldots,t_n\}} \sqrt{\textstyle\sum_{j=1}^{d}(V(t)[j] - V'(t)[j])^2} \tag{23}$$

Consider the j-th dimension. We denote the SAD error between the projections of T and that of T' in the j-th dimension by $\epsilon_{SAD}(T'|T)[j]$.

$$\epsilon_{SAD}(T'|T)[j] = \max_{t\in\{t_1,t_2,\ldots,t_n\}} \sqrt{(V(t)[j] - V'(t)[j])^2} \tag{24}$$

Then, it holds that $\epsilon_{SAD}(T'|T)$ is at most δ if for each dimension j, $\epsilon_{SAD}(T'|T)[j] \leq \frac{\delta}{\sqrt{d}}$, where d is the total number of dimensions.

Proof. Consider the SAD between a raw trajectory T and its simplified trajectory T'.

$$\begin{aligned}
&\epsilon_{SAD}(T'|T)\\
=&\max_{t\in[t_1,t_n]}\sqrt{\sum_{j=1}^{d}(V(t)[j]-V'(t)[j])^2}\\
\leq&\max_{t\in[t_1,t_n]}\sqrt{d\cdot\max_{j\in[1,d]}(V(t)[j]-V'(t)[j])^2}\\
=&\sqrt{d}\max_{t\in[t_1,t_n]}\sqrt{\max_{j\in[1,d]}(V(t)[j]-V'(t)[j])^2}\\
=&\sqrt{d}\max_{t\in[t_1,t_n]}\max_{j\in[1,d]}\sqrt{(V(t)[j]-V'(t)[j])^2}\\
=&\sqrt{d}\max_{j\in[1,d]}\max_{t\in[t_1,t_n]}\sqrt{(V(t)[j]-V'(t)[j])^2}\\
=&\sqrt{d}\max_{j\in[1,d]}\epsilon_{SAD}(T'|T)[j]
\end{aligned} \tag{25}$$

From Inequality (25), we obtain that if $\epsilon_{SAD}(T'|T)[j] \leq \frac{\delta}{\sqrt{d}}$ for any dimension j, $\epsilon_{SAD}(T'|T)$ must be less than or equal to δ.

Theorem 3. *Let $T'_1, T'_2, T'_3, ..., T'_d$ denote the input of our interpolation algorithm (each T'_j, $j \in [1, d]$ is a one-dimensional trajectory simplified from the projection of T on the j-th dimension). For better clarity, we denote the coordinate value at timestamp t of T'_j by $T'_j(t)$. Let T' denote the output of our interpolation algorithm. Then, it holds that*

$$T'(t)[j] = T'_j(t) \tag{26}$$

That is, the projection of T' on the j-th dimension coincides with T'_j. Besides, it holds that

(1) *$\epsilon_{SED}(T'|T)$ is at most δ if for each dimension j, $\epsilon(T'_j|T_j)_{SED} \leq \frac{\delta}{\sqrt{d}}$, where d is the total number of dimensions.*
(2) *$\epsilon_{SAD}(T'|T)$ is at most δ if for each dimension j, $\epsilon(T'_j|T_j)_{SAD} \leq \frac{\delta}{\sqrt{d}}$, where d is the total number of dimensions.*

Proof. We first present the two lemmas as follows.

Lemma 1. *The value ω in Line 9 of Algorithm 3 is equal to $T'_j(t_k)$.*

Proof. According to Equation (1), we obtain that $T'_j(t) = \frac{T'_j(t_{\tau+1})-T'_j(t_\tau)}{t_{\tau+1}-t_\tau}\cdot(t-t_\tau)$, where $t \in [t_\tau, t_{\tau+1}]$. Thus, we obtain that $T'_j(t_k) = \frac{T'_j(t_{\tau+1})-T'_j(t_\tau)}{t_{\tau+1}-t_\tau}\cdot(t-t_\tau)$ which we assign to ω in Line 9 of Algorithm 3.

Lemma 2. *The newly created point p' in Line 14 of Algorithm 3 lies on the line segment $\overline{T'(t_k)T'(t_{k+1})}$ and the j'-th coordinate value of p' is equal to $T'[j'](t)$ $\forall j' \in [1, j-1]$.*

Proof. According to Equation (1), we obtain that $T'(t) = \frac{T'(t_{k+1})-T'(t_k)}{t_{k+1}-t_k}\cdot(t-t_k)$, where $t \in [t_k, t_{k+1}]$, which we assign to p' in Line 14 of Algorithm 3. Thus, we obtain that the point p' lies on the line segment $\overline{T'(t_k)T'(t_k)}$.

By the two lemmas above, we obtain that

$$T'(t)[j] = T'_j(t) \tag{27}$$

By this equation and Theorem 1 and Theorem 2, we obtain that

(1) $\epsilon_{SED}(T'|T)$ is at most δ if for each dimension j, $\epsilon(T'_j|T_j)_{SED} \leq \frac{\delta}{\sqrt{d}}$, where d is the total number of dimensions.
(2) $\epsilon_{SAD}(T'|T)$ is at most δ if for each dimension j, $\epsilon(T'_j|T_j)_{SAD} \leq \frac{\delta}{\sqrt{d}}$, where d is the total number of dimensions.

Thus, we further obtain the following two theorems.

Theorem 4. *Let $T'_1, T'_2, T'_3, ..., T'_d$ denote the input of our interpolation algorithm (each $T'_j, j \in [1, d]$, is a one-dimensional trajectory simplified from the projection of T on the j-th dimension). For better clarity, we denote the coordinate value at timestamp t of T'_j by $T'_j(t)$. Let T' denote the output of our interpolation algorithm. Then, it holds that the size $|T'|$ is at most $\sum_{j\in[1,d]} |T'_j|$.*

Proof. We prove this theorem by induction. Consider the j-th dimension, where $j \in [1, d]$. It is trivially true that initially $|T'| \leq |T'_1|$ when we only consider the first dimension (i.e., $j = 1$) since in Line 1 of Algorithm 3, T' is assigned to be T'_1. In the induction, we assume that in the $j-1$ iterations in the for-loop of Algorithm 3, $|T'| \leq \sum_{j'=1}^{j-1} |T'_{j'}|$. Then, consider the j-th iteration in the for-loop. In Lines 3-11, we do not insert any new point into T' and thus, the size of T' keeps intact. In the inner for-loop in Lines 12-19, we inserts $|T'_j|$ points into T'. Thus, we obtain that after j-th iteration, the size of T' is at most $\sum_{j'=1}^{j} |T'_{j'}|$.

Lemma 3. *The size of each one-dimensional simplified trajectory T'_j in the output of Algorithm 4 contains exactly W points.*

Proof. According to Lines 1-5 and Lines 11-12 of Algorithm 4, the size of the buffer $\mathcal{B}^i, \forall i \in [1, n]$ is equal to W. Thus, the output which is assigned to be $\mathcal{B}^n$ has exactly W points.

Theorem 5. *Let T' denote the output of our algorithm (Algorithm 1) for Min-Error problem (consisting of the one-dimensional simplification and interpolation). It holds that $|T'| \leq \theta$, where θ is the budget for the Error-Bounded problem (Algorithm 1). The error function $\epsilon(\cdot|\cdot)$ can be SED or SAD which is specified in the one-dimensional simplification.*

Proof. According to Algorithm 1, the budget W for each one-dimensional trajectory simplification algorithm (Algorithm 4) is set to be $\frac{\theta}{d}$. By Lemma 3, the size of each one-dimensional simplified $T'_i, i \in [1, d]$ is at most $\frac{\theta}{d}$ in the input of the interpolation algorithm in Algorithm 1. By Theorem 4, we obtain that the size of the final output T' is at most $\sum_{j=1}^{d} T'_j \leq \theta$.

Lemma 4. *It hold that for each one-dimensional simplified trajectory T'_j in the output of our algorithm (Algorithm 5) for Error-bounded problem, the error $\epsilon(T'_j|T_j)$ is at most ϵ, where ϵ is the error parameter of Algorithm 5.*

Proof. We prove this lemma by contradiction. Assume that the one-dimensional simplified trajectory T'_j in the output of our algorithm (Algorithm 5) for Error-bounded problem has its error $\epsilon(T'_j|T_j)$ larger than ϵ, where ϵ is the error parameter of Algorithm 5. Then, there must be a point $T_j(t_i)$ in T_j such that $\epsilon(T'_j(t_i)|T_j(t_i)) > \epsilon$, where i is in the range of $[1, n]$, let t_{s_k} denote the timestamp of one sampled point in T'_j such that $t_{s_k} \leq t_i < t_{s_k+1}$. Let $T_j(t_x)$ and $T_j(t_y)$ denote the points in T_j with the same timestamp with $T'_j(t_{s_k})$ and $T'_j(t_{s_k+1})$, respectively. According to Line 4, Line 12 and Line 14 in Algorithm 5, the points $T_j[t_x]$ and $T_j[t_y]$ are inserted into T'_j only if $\epsilon(\overline{T_j(t_x)T_j(t_y)}|T_j(t_x : t_y)) \leq \epsilon$. But our assumption implies that $\epsilon(T'_j(t_i)|T_j(t_i)) > \epsilon$, where $t_x \leq t_i \leq t_y$,

which further implies that $\epsilon(\overline{T_j(t_x)T_j(t_y)}|T_j(t_x : t_y)) > \epsilon$ since $T'_j(t_i)$ lies on the line segment $\overline{T_j(t_x)T_j(t_y)}$. Contradiction!

Theorem 6. *Let T' denote the output of our algorithm (Algorithm 2) for Error-Bounded problem (consisting of the one-dimensional simplification and interpolation). It holds that $\epsilon(T'|T) \leq \delta$, where δ is the error parameter for the Error-Bounded problem. The error function $\epsilon(\cdot|\cdot)$ can be SED or SAD which is specified in the one-dimensional simplification.*

Proof. By Algorithm 2, the error parameter ϵ for each one-dimensional trajectory simplification algorithm is set to be $\frac{\delta}{\sqrt{d}}$. Thus, each one-dimensional simplified trajectory in the input of the interpolation algorithm in Algorithm 2, denoted by $T_j, j \in [1, d]$, has an error within $\frac{\delta}{\sqrt{d}}$ compared with T_i (i.e., $\epsilon(T'_i|T_i) \leq \frac{\delta}{\sqrt{d}}$). Then, by Theorem 3, we obtain that the final error $\epsilon(T'|T)$ is at most δ.

Lemma 5. *The time complexity of our interpolation algorithm (Algorithm 3) is $O(d^2 \cdot n + d \cdot n \log n)$.*

Proof. Consider the outer for-loop in Lines 2-23. There are d iterations in this outer for-loop. Consider the inner for-loop in Lines 4-20. There are at most n iterations in this inner for-loop. In the inner for-loop, we consider each point in the currently maintained T' only once and we also consider each point in T'_j only once. For each point in T', it takes $O(j-1) = O(d)$ times for the computation in Lines 5-9 and for each point in T'_j, it takes $O(j-1) = O(d)$ times for the computation in Lines 13-18. Thus, the inner for-loop takes $O(d \cdot n)$ time. Since it takes $O(n \log n)$ time to sort all points in $\mathcal{T}$ in line 21 (which is outside the inner for-loop but within the outer for-loop), the outer for-loop takes $O(d^2 n + dn \log n)$ time.

Theorem 7. *The time complexity of the algorithm Simp-ME is $O(d^2 \cdot n + d \cdot n \log n + d(n-\theta) \cdot \psi + d\theta \log \theta)$, where ψ is the cost of computing the value of a point.*

Proof. Simp-ME consists of d 1D-ME algorithms (Algorithm 4) and the Interpolation algorithm (Algorithm 3). Consider 1D-ME algorithm (Algorithm 4). The for-loop in Lines 3-5 takes $O(\theta)$ time. The for-loop in Lines 6-13 takes $O((n-\theta) \cdot \psi)$ time and it takes $O(\theta \log \theta)$ to sort all points in $\mathcal{B}$ in Line 15. Thus, we conclude that the time complexity of 1D-ME algorithm (Algorithm 4) is $O((n-\theta) \cdot \psi + \theta \log \theta)$. By Lemma 5, the time complexity of our interpolation algorithm (Algorithm 3) is $O(d^2 \cdot n + d \cdot n \log n)$. Finally, we obtain that the time complexity of Simp-ME is $O(d^2 \cdot n + d \cdot n \log n + d(n-\theta) \cdot \psi + d\theta \log \theta)$.

Theorem 8. *The time complexity of the algorithm Simp-EB is $O(d^2 \cdot n + d \cdot n \log n + dn \cdot \psi)$, where ψ is the cost of computing the value of a point.*

Proof. Simp-EB consists of d 1D-EB algorithms (Algorithm 5) and the Interpolation algorithm (Algorithm 3). Consider 1D-EB algorithm (Algorithm 5). The for-loop in Lines 3-5 takes $O(n)$ time. The for-loop in Lines 6-15 takes $O((n) \cdot \psi)$ time. Thus, we conclude that the time complexity of 1D-ME algorithm (Algorithm 4) is $O(n \cdot \phi)$.

By Lemma 5, the time complexity of our interpolation algorithm (Algorithm 3) is $O(d^2 \cdot n + d \cdot n \log n)$. Finally, we obtain that the time complexity of Simp-ME is $O(d^2 \cdot n + d \cdot n \log n + dn \cdot \psi)$.

4 Experiment

4.1 Experimental Setting

Evaluation Platform. The implementation of all methods is carried out in Python 3.8 on Ubuntu 22.04. Specifically, our implementations are based on TensorFlow 2.5.0. The experiments are conducted on a machine

equipped with an Intel(R) 12 vCPU Intel(R) Xeon(R) Silver 4214R CPU operating at 2.40GHz, 90.0GB RAM, and a Nvidia GeForce GTX 3080 GPU.

Datasets. Our experimental evaluations are conducted on three real-world trajectory datasets, namely Geolife [45] and MOPSI [22]. The Geolife dataset encompasses the outdoor trajectories of 182 users over a span of five years. On the other hand, MOPSI is a subset of the comprehensive Mopsi dataset, comprising 6,779 routes (7,850,387 points) recorded by 51 users. We also generated three datasets—Dialogue-4D, Dialogue-5D, and Dialogue-6D—by applying the ActiveRanking algorithm [9] to several dialogue datasets [34, 35] with 4, 5, and 6 attributes, respectively. This algorithm is commonly used in chatbots and dialogue systems. During the dialogue process, multiple attributes are considered, each attribute representing a distinct dimension. As a result, each dialogue state can be represented as a point (i.e., a concatenation of these attributes), and the entire sequence of multi-round dialogues can be modeled as a trajectory. The information of the trajectory datasets is detailed in Table 3.

Dataset	Dim.	No. of Trajectories	No. of Points	Domain	Reference
GeoLife	2	17, 621	24,876,978	Taxi Trajectories	[45]
MOPSI	3	2, 020	1,937,912	Animal Movement	[22]
Dialgue-4D	4	15,594	5,168,199	Dialogue System & Chatbots	[9]
Dialgue-5D	5	15, 274	2,774,191	Dialogue System & Chatbots	[9]
Dialgue-6D	6	14, 694	4,926,386	Dialogue System & Chatbots	[9]

Table 3. Statistics of Trajectory Datasets (Dim. stands for Dimensionality)

Error Functions. In this experiment, we adopted the *Synchronized Euclidean Distance* (*SED*) and the *Synchronized Angular Distance* (*SAD*) to measure the simplification error for a simplified trajectory T w.r.t. the original trajectory T.

Parameter Setting and Network Configuration. Our methods employ a neural network architecture consisting of three layers: an input layer, a hidden layer, and an output layer. The hidden layer comprises 20 neurons and utilizes the hyperbolic tangent (tanh) function as the activation function. To address data scaling concerns, batch normalization from TensorFlow is applied prior to activation. In our approach, the output layer consists of k neurons, where the default value of k is set to 3. From the training dataset, we randomly select 1,000 trajectories, generating 10 episodes per trajectory for policy learning. Considering that each trajectory encompasses approximately 1,000 data points, the learning process involves approximately 10 million transition steps.

The optimization process is performed using the Adam stochastic gradient descent algorithm with a learning rate of 0.001, determined empirically. Various settings for the reward discount factor were experimented with, and due to consistent outcomes across trials, it is fixed at 0.99. The policy that yields the highest reward per episode is selected for trajectory simplification.

4.2 Experimental Results

Figure 3 and Figure 4 demonstrates the experimental results of Min-Error problem under SED and SAD error functions, respectively. In this experiment, we compare our proposed algorithm for the Min-Error problem, namely *HITS-ME* with the state-of-the-art algorithm *RLTS* [36]. Since RLTS can only be applied to 2D trajectories,

there are no results of RLTS shown in higher dimensional datasets. Besides, we also compare our algorithm with a baseline called *Random* (*Min-Error*) which uniformly samples W points from the given raw trajectory at random. Following the existing study [36], the storage budgets considered in the experiment vary from $0.1 \cdot |T|$ to $0.5 \cdot |T|$. As shown therein, our algorithm HITS-ME achieves comparable simplification error with the state-of-the-art algorithm in the 2D trajectory dataset *GeoLife*. Moreover, it significantly outperforms the competitors regarding the simplification error in the high-dimensional trajectory datasets including MOPSI (3D), Dialogue-4D, Dialogue-5D, and Dialogue-6D.

Figure 5 and Figure 6 demonstrates the experimental results of Error-Bounded problem under SED and SAD error functions, respectively. In this experiment, we compare our proposed algorithm for Error-Bounded problem, namely *HITS-EB* with the state-of-the-art algorithm *MARL4TS* [39]. Since MARL4TS can only be applied to 2D trajectories, there are no results of MARL4TS shown in higher dimensional datasets. Besides, we also compare our algorithm with two other baselines called *Random* (*Error-Bounded*) and *No-Error*. *Random* (*Error-Bounded*) works as follows. Given a raw trajectory T, it initializes the simplified trajectory T' to be $\emptyset$. Then, it iteratively selects one randomly selected point and inserts it into T'. It terminates and returns T' once the error $\epsilon(T'|T)$ is smaller than δ, where the error function $\epsilon(|)$ can be SED or SAD and δ is the user-specified error parameter. *No-Error* linearly scans each point $T(t_i)$ in T and removes it if $T(t_i)$ lies on the line segment $\overline{T(t_{i-1})T(t_{i+1})}$ for each $i \in [2, n-1]$ (that is, the removal of $T(t_i)$ will not incur any simplification error). Thus, the simplified trajectory T' returned by *No-Error* has zero simplification error and the results of this algorithm demonstrate the best compression ratio with zero simplification error. As the figures show, our algorithm achieves comparable simplification compression ratio with the state-of-the-art algorithm in the 2D trajectory dataset *GeoLife*, and our algorithm significantly outperforms the competitor in terms of the compression ratio in the high-dimensional trajectory datasets including MOPSI (3D), Dialogue-4D, Dialogue-5D, and Dialogue-6D. Besides, the *No-Error* algorithm has the highest compression ratio (which is very close to 1) and this implies that this naive simplification algorithm fails to effectively compress the trajectory.

Figure 7 presents the results of our ablation study conducted on the MOPSI dataset. Figures 7(a) and 7(b) illustrate the error of our algorithm, HITS-ME, as the storage budget varies between $0.1 \cdot |T|$ and $0.5 \cdot |T|$. In these figures, we capture two parts of results: (1) the simplification error for each dimension (labeled as Dim 1, Dim 2, and Dim 3) and (2) the total simplification error (labeled as All), which reflects the total error after interpolation. The findings indicate that the simplification error remains small across each dimension, and the overall error remains low after interpolation. Figures7(c) and 7(d) show the performance of our HITS-EB algorithm under different error tolerances. We recorded: (1) the compression ratio for each dimension (shown as Dim 1, Dim 2, and Dim 3) and (2) the overall compression ratio (labeled as All) that represents the total compression achieved after interpolation. These results demonstrate that the compression ratio was consistently low for each dimension, and the total compression ratio remained minimal following the interpolation process.

5 Related Work

5.1 Traditional Rule-Based Two Dimensional Trajectory Simplification Algorithms

Among the existing studies on the rule-based trajectory simplification, [14, 25, 26, 29] target the Min-Error problem. [1] proposed a dynamic programming algorithm for the Min-Error problem. It takes at least $O(n^3)$ time for compressing a trajectory with n points which is prohibitively expensive. Motivated by this, a lot of

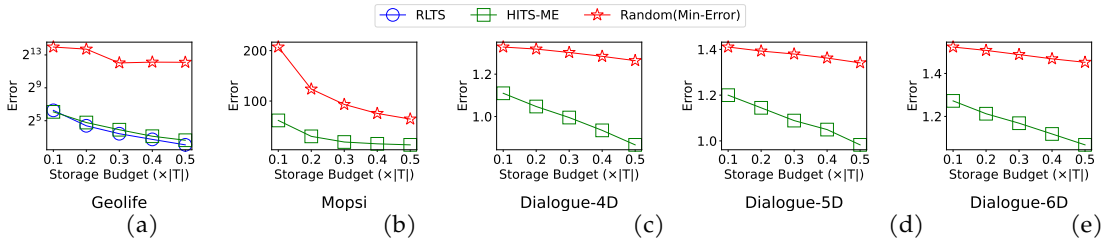

Fig. 3. Min-Error Problem (SED)

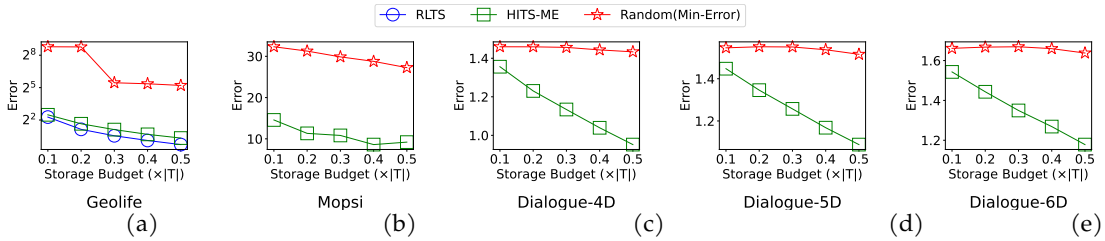

Fig. 4. Min-Error Problem (SAD)

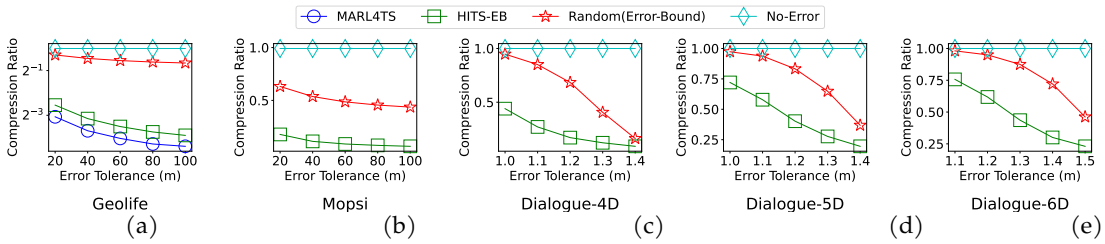

Fig. 5. Error-Bound Problem (SED)

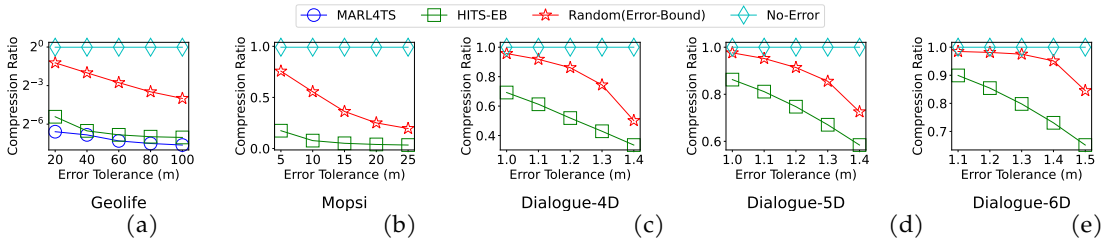

Fig. 6. Error-Bound Problem (SAD)

subsequent studies [12, 14, 20, 23, 25, 26, 29] focuses on the design of approximation algorithms with better execution efficiency. Each of them adopted a hand-crafted greedy algorithm to either (i) remove points one by one from the original trajectory or (ii) starting from the first and the last points in the original trajectory, include points one by one. Note that the greedy algorithm that each one developed is only applied for the specific error function that it targets for. We also refer the readers to the survey [43] for more detailed presentation and a

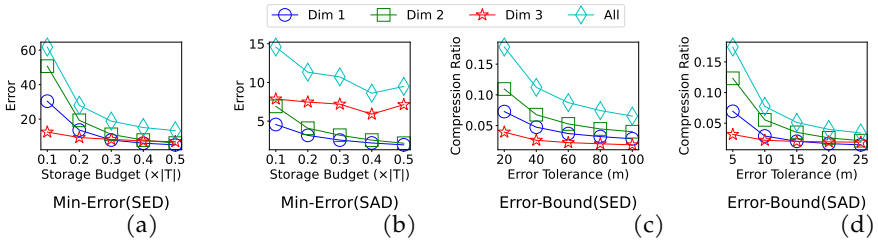

Fig. 7. Ablation (Mopsi)

systematical empirical comparison of the algorithms above. Many existing algorithms have been developed for the Error-Bounded problem, including [2, 4, 5, 10, 11, 16–19, 21, 23]. All except DOTS adopt the three-step framework in which Step 1 opens a window containing two adjacent points in the trajectory, Step 2 expand the window by checking the subsequent points one by one and see if the removal of each violates the error constraint But use different strategies for expanding and Step 3 store the first point which violates the constraint and re-open a new window starting from this point. All algorithms in this category are traditional rule-based algorithms. Their algorithms are based on the hand-crated rules designed for a specific error function or several specific functions which highly prevent their usage in many application where the error measurements are different. Besides, they can not be generalized to the generic error function scenarios where there is a novel error measurement considered and their rules are limited to 2D trajectories.

5.2 Deep Reinforcement Learning-Based Two Dimensional Trajectory Simplification Algorithms

The algorithms in this category reformulate the online trajectory compression into a reinforcement learning problem. In the reinforcement learning paradigm, there are two components, namely *agent* and *environment*, and the agent tries to obtain as large accumulative rewards as possible through the interaction with the environment. Specifically, the environment consists of *states*, *rewards*, *actions*, *transitions* and a *decay factor*. The states defines all possible statuses that the agent stays, the actions provides the choices that the agent could take in each state. Transitions defines how the next state that the agent transits to after an action is taken. For each pair of state and action, there is a reward r given to the agent and the intermediate reward obtained is calculated as $r \times d^t$, where d denotes the decay factor and t denote the number of the steps that the agent takes from the beginning. Finally, the accumulative reward is the sum of all intermediate rewards obtained in all steps. A policy is defined as a mapping from each state to an action. It is worth mentioning that given a fixed policy, the environment is a markov decision process (MDP). From the description above, we could observe that the task of the agent is to learn an optimal policy which provides the optimal action given the current state such that the accumulative rewards could be maximized. The first attempt of the existing works [36] which applies the reinforcement learning to the online trajectory compression works as follows and it focuses on Min-Error Problem. The setting of the agent and environment of RL is reformulated in the context of 2D trajectory simplification. In their problem setting, the actions in each step consists of keeping the current point and dropping it. The reward is defined as the error decrease after the corresponding action is taken. The state contains several indicators extracted from the currently maintained partial trajectory based on the partial trajectory which could be seen in the current step. The transition updates the indicators accordingly

after each action. Similarly, [39] proposes using a multi-agent deep reinforcement learning method to tackle Error-Bounded Problem and one of the agents determines whether the current point should be included in T' and opens a new window if it determines to include it and the other agent detects all the redundant points which could be safely discarded without violating the error constraint in the current window. The handcrafted rules are designed for their states, rewards, transitions and actions. After that, [37] proposed an RL-based algorithm to collectively simplified a 2D trajectory dataset instead of each individual 2D trajectory. Then, [6] proposed an sequence-to-sequence-to-sequence (S3) framework for the 2D trajectory simplification. It adopts a neural compressor to simplify the trajectory and utilizes a neural re-constructor to calculate the reconstruction loss of the simplified trajectory w.r.t. the raw trajectory. The whole framework can be trained in an end-to-end fashion. But as mentioned before, the category of algorithms is limited to 2D trajectories since their design of the rules or the states and rewards in RL highly relies on the 2D geometries and can not be applied to high-dimensional trajectories.

5.3 High-Dimensional Trajectory Simplification Algorithm

As far as we are concerned, there is only one existing work on high-dimensional trajectory simplification [7]. It first proposed an error function called L-∞ *Euclidean Distance* for measuring the simplification error of a simplified trajectory w.r.t. the raw trajectory. This error function has not been adopted in any other work yet and it lacks the intuition and motivation and has not found any application scenario. Then, the work developed a set of handcrafted rules which is only limited to this L-∞ Euclidean Distance for the high-dimensional trajectory simplification and can not be applied to general error function.

5.4 Other Related Study

This section reviews some other relevant studies. [3, 8, 15, 31, 41] studies the road network-based trajectory compression problem. In their problem setting, they assume that there is a road network associated with the set of trajectories. Their goal is to match each point in a trajectory back to the corresponding road segment and then further reduce the complexity of the trajectory by using the network topology. But they are limited to the application scenarios where a road network must be given and can not be applied to the trajectories in a free space (e.g., pedestrian trajectories in open areas [46], animal trajectories in the wildness [19], sports players' trajectories in the playground [38, 44], etc.) which our method are capable of. There are also studies [13, 29, 33] on the efficiently online sampling of trajectories which online discards some points sampled based on many different criteria by using the dead reckoning.

6 Conclusion

In this paper, we propose to study the high-dimensional trajectory simplification problem which finds wide applications in the urban computing, smart city, dialogue systems, and etc. We observe that most existing trajectory simplification algorithms can only applied to 2D trajectories and the only high-dimensional trajectory simplification algorithm is limited to $L-\infty$ error metric which is not practical and never considered in other works. Motivated by this, we propose a reinforcement learning-based algorithm which can be widely applied to any simplification error metric and can scale up to high dimensions. Our algorithm first projects the raw d-dimension ($d \geq 2$) trajectory into d one-dimensional trajectories. Then, we apply an RL agent to simplify each one-dimensional trajectories and finally, we merge the d simplified one-dimensional trajectories into a

simplified d-dimensional trajectory through interpolation. We theoretically analyze the relationships between the size (resp. error) of each simplified one-dimensional trajectory and that of the final simplified d-dimensional trajectory. Based upon this theoretical results, we properly set the hyperparameters of our RL agent to guarantee the size or the error of the final simplified d-dimensional trajectory. Our experimental results show that our algorithm is comparable to the best existing algorithms for the 2D trajectory simplification and significantly outperforms all baselines for high-dimensional trajectory simplifications. In the future, we plan to also study the collective high-dimensional trajectory simplification problem which simplifies all trajectories collectively in a dataset.

References

[1] R. Bellman. On the approximation of curves by line segments using dynamic programming. *Communications of the ACM*, 4(6):284, 1961.

[2] W. Cao and Y. Li. Dots: An online and near-optimal trajectory simplification algorithm. *Journal of Systems and Software*, 2017.

[3] C. Chen, Y. Ding, X. Xie, S. Zhang, Z. Wang, and L. Feng. Trajcompressor: An online map-matching-based trajectory compression framework leveraging vehicle heading direction and change. *TITS*, 2019.

[4] M. Chen, M. Xu, and P. Franti. A fast $o(n)$ multiresolution polygonal approximation algorithm for gps trajectory simplification. *IEEE Transactions on Image Processing*, 2012.

[5] Y. Chen, K. Jiang, Y. Zheng, C. Li, and N. Yu. Trajectory simplification method for location-based social networking services. In *Proceedings of the 2009 international workshop on location based social networks*, pages 33–40, 2009.

[6] Z. Fang, C. He, L. Chen, D. Hu, Q. Sun, L. Li, and Y. Gao. A lightweight framework for fast trajectory simplification. In *ICDE*, pages 2386–2399, 2023.

[7] Y. Han and H. Samet. Limits: An effective approach for trajectory simplification. *arXiv preprint arXiv:2010.08622*, 2020.

[8] Y. Han, W. Sun, and B. Zheng. Compress: A comprehensive framework of trajectory compression in road networks. *ACM Transactions on Database Systems*, 42(2):1–49, 2017.

[9] K. G. Jamieson and R. D. Nowak. Active ranking using pairwise comparisons. In *NIPS*, 2011.

[10] B. Ke, J. Shao, and D. Zhang. An efficient online approach for direction-preserving trajectory simplification with interval bounds. In *2017 18th IEEE International Conference on Mobile Data Management*, pages 50–55. IEEE, 2017.

[11] B. Ke, J. Shao, Y. Zhang, D. Zhang, and Y. Yang. An online approach for direction-based trajectory compression with error bound guarantee. In *Asia-Pacific Web Conference*. Springer, 2016.

[12] E. Keogh, S. Chu, D. Hart, and M. Pazzani. An online algorithm for segmenting time series. In *Proceedings 2001 IEEE international conference on data mining*, pages 289–296. IEEE, 2001.

[13] R. Lange, F. Dürr, and K. Rothermel. Efficient real-time trajectory tracking. *The VLDB Journal*, 20:671–694, 2011.

[14] H. Li, L. Kulik, and K. Ramamohanarao. Spatio-temporal trajectory simplification for inferring travel paths. In *SIGSPATIAL*, pages 63–72, 2014.

[15] T. Li, R. Huang, L. Chen, C. S. Jensen, and T. B. Pedersen. Compression of uncertain trajectories in road networks. *VLDB*, 2020.

[16] X. Lin, J. Jiang, S. Ma, Y. Zuo, and C. Hu. One-pass trajectory simplification using the synchronous euclidean distance. *The VLDB Journal*, 28(6):897–921, 2019.

[17] 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*, pages 987–998, 2015.

[18] 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. *TKDE*, 28(11):2827–2841, 2016.

[19] C. Long, R. C.-W. Wong, and H. Jagadish. Direction-preserving trajectory simplification. *VLDB*, 6(10):949–960, 2013.

[20] C. Long, R. C.-W. Wong, and H. Jagadish. Trajectory simplification: On minimizing the direction-based error. *VLDB*, 8(1):49–60, 2014.

[21] X. L. S. Ma and H. Z. T. W. J. Huai. One-pass error bounded trajectory simplification. *VLDB*, 10(7), 2017.

[22] R. Mariescu-Istodor and P. Fränti. Grid-based method for gps route analysis for retrieval. *ACM Transactions on Spatial Algorithms and Systems (TSAS)*, 3(3):1–28, 2017.

[23] N. Meratnia and A. Rolf. Spatiotemporal compression techniques for moving point objects. In *International Conference on Extending Database Technology*, pages 765–782. Springer, 2004.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.

[25] J. Muckell, J.-H. Hwang, V. Patil, C. T. Lawson, F. Ping, and S. Ravi. Squish: an online approach for gps trajectory compression. In *Proceedings of the 2nd international conference on computing for geospatial research & applications*, pages 1–8, 2011.

[26] J. Muckell, P. W. Olsen, J.-H. Hwang, C. T. Lawson, and S. Ravi. Compression of trajectory data: a comprehensive evaluation and new approach. *GeoInformatica*, 18(3):435–460, 2014.
[27] J. Muckell, P. W. Olsen, J.-H. Hwang, C. T. Lawson, and S. Ravi. Compression of trajectory data: a comprehensive evaluation and new approach. *GeoInformatica*, 18(3):435–460, 2014.
[28] L. Po, F. Rollo, J. R. R. Viqueira, R. T. Lado, A. Bigi, J. C. López, M. Paolucci, and P. Nesi. Trafair: Understanding traffic flow to improve air quality. In *2019 IEEE International Smart Cities Conference (ISC2)*, 2019.
[29] M. Potamias, K. Patroumpas, and T. Sellis. Sampling trajectory streams with spatiotemporal criteria. In *18th International Conference on Scientific and Statistical Database Management*, 2006.
[30] M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.
[31] R. Song, W. Sun, B. Zheng, and Y. Zheng. Press: A novel framework of trajectory compression in road networks. *VLDB*, 2014.
[32] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*. MIT press, 2018.
[33] 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 *Proceedings of the 5th ACM international workshop on Data engineering for wireless and mobile access*, 2006.
[34] W. Wang, R. C.-W. Wong, and M. Xie. Interactive search for one of the top-k. In *SIGMOD*, New York, NY, USA, 2021. ACM.
[35] W. Wang, R. C.-W. Wong, and M. Xie. Interactive search with mixed attributes. In *ICDE*, 2023.
[36] Z. Wang, C. Long, and G. Cong. Trajectory simplification with reinforcement learning. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 684–695. IEEE, 2021.
[37] Z. Wang, C. Long, G. Cong, and C. S. Jensen. Collectively simplifying trajectories in a database: A query accuracy driven approach. In *ICDE*, pages 4383–4395, 2024.
[38] Z. Wang, C. Long, G. Cong, and C. Ju. Effective and efficient sports play retrieval with deep representation learning. In *SIGKDD*, pages 499–509, 2019.
[39] Z. Wang, C. Long, G. Cong, and Q. Zhang. Error-bounded online trajectory simplification with multi-agent reinforcement learning. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1758–1768, 2021.
[40] L. Xia and Y. Shao. Modelling of traffic flow and air pollution emission with application to hong kong island. *Environmental Modelling & Software*, 20(9):1175–1188, 2005.
[41] X. Yang, B. Wang, K. Yang, C. Liu, and B. Zheng. A novel representation and compression for queries on trajectories in road networks. *TKDE*, 30(4), 2017.
[42] A. Zaldei, F. Camilli, T. De Filippis, F. Di Gennaro, S. Di Lonardo, F. Dini, B. Gioli, G. Gualtieri, A. Matese, W. Nunziati, L. Rocchi, P. Toscano, and C. Vagnoli. An integrated low-cost road traffic and air pollution monitoring platform for next citizen observatories. *Transportation Research Procedia*, 24:531–538, 2017.
[43] D. Zhang, M. Ding, D. Yang, Y. Liu, J. Fan, and H. T. Shen. Trajectory simplification: an experimental study and quality analysis. *VLDB*, 2018.
[44] Q. Zhang, Z. Wang, C. Long, and S.-M. Yiu. On predicting and generating a good break shot in billiards sports. In *Proceedings of the 2022 SIAM International Conference on Data Mining*, 2022.
[45] Y. Zheng, X. Xie, W.-Y. Ma, et al. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, 33(2):32–39, 2010.
[46] Y. Zheng, X. Xie, W.-Y. Ma, et al. Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, 33(2):32–39, 2010.
[47] J. Ángel Martín-Baos, L. Rodriguez-Benitez, R. García-Ródenas, and J. Liu. Iot based monitoring of air quality and traffic using regression analysis. *Applied Soft Computing*, 115, 2022.