# Network Trajectory Protection

Qianting Chen[1,2], Lulu Wang[3], Ke Chao[3], Weicheng Wang[4], Rui Mao[1]
[1]Shenzhen University, Shenzhen, China
[2]University of Macau, Macau, China
[3]Beijing Normal University, Beijing, China
[4]The Chinese University of Hong Kong, Hong Kong, China

**In modern computer networks, the transmission of messages forms a sequence through multiple intermediate nodes, which can be represented as network trajectories. These trajectories provide essential information for routing optimization and security auditing. However, the former requires learning the clear trajectory, while the latter needs to hide the details of the trajectory. There is a contradiction between the need for accurate trajectory analysis and the need to conceal sensitive details. To balance these objectives, trajectory simplification becomes essential: reducing the number of recorded nodes while preserving the overall structure. Yet, existing simplification algorithms primarily target trajectories with nodes described by 2-dimensional vectors and are inadequate for multi-dimensional settings. To address this, we propose the Network Trajectory Protection (NTP) problem: given a multi-dimensional trajectory and a simplification budget, the goal is to select a subset of nodes that best approximates the original trajectory under a Euclidean distance-based loss metric. We prove that the NTP problem is NP-hard and design an efficient greedy approximation algorithm. We conduct extensive experiments on network trajectory datasets, demonstrating the superior effectiveness and efficiency of our algorithms compared to baselines.**

*Index Terms*—**Network Trajectory, Protection, Trajectory Simplification**

## I. Introduction

Trajectory is commonly used in many scenarios. In modern computer networks, when a message is sent, it travels through a sequence of intermediate nodes such as routers, switches, and gateways before arriving at the final destination. This transmission forms a *trajectory* that consists of multiple intermediate nodes. For example, consider a message that originates from a data center in Los Angeles and is destined for a user in Sydney, as shown in Figure 1. The message may travel through the nodes in New York, Paris, Beijing, and ultimately reach Sydney. The intermediate nodes traversed can be recorded as a trajectory [New York, Paris, Beijing]. In smart city infrastructures [1]–[3], data flow from multi-source, heterogeneous sensors—such as traffic cameras and environmental monitoring stations—form trajectories. More broadly, in the Internet of Things (IoT) [4], [5], trajectories naturally emerge in diverse applications: smart transportation systems record the paths of connected vehicles and public transit flows; industrial IoT platforms monitor the trajectory of goods and components across manufacturing pipelines and supply chains; and agricultural IoT networks capture the trajectories of drones and autonomous farming machines for precision agriculture.

The trajectories are not only vital for routing optimization, congestion management, etc, but also serve as an essential layer of observability for security auditing. However, there is a contradiction in serving these two parts of the goals. The former requires learning the clear trajectory, while the latter needs to hide the details of the trajectory. In this case, there is an issue raised: *how to handle the network trajectory?*

To mitigate this issue, it is often desirable to simplify the trajectories while preserving their essential structure. Here, the network trajectory simplification refers to reducing the number of nodes in the trajectory such that the resulting simplified trajectory remains a close approximation of the original trajectory. The simplified representation enables routing optimization, congestion management, etc, while also offering a degree of obfuscation that can help mitigate privacy and security risks associated with full exposure. For example, consider the network trajectory in Figure 1, a simplified version might retain only the two end nodes: [New York, Beijing]. The simplification implicitly assumes that the omitted nodes do not deviate significantly from the preserved trajectory in terms of semantic distance.

Existing studies in trajectory simplification predominantly focused on the nodes described by 2-dimensional vectors, such as GPS trajectories of vehicles or individuals moving in geographic space. A large body of work [6]–[9] proposes heuristic or rule-based algorithms for simplifying such 2-dimensional trajectories. These algorithms mainly act in two ways: (1) the bottom-up manner (iteratively removing the least significant node) and (2) the top-down manner (recursively dividing the trajectory into segments), often based on hand-crafted rules tailored to specific loss metrics.

However, existing algorithms face serious limitations when applied to trajectories in higher-dimensional spaces, such as network information flows. Network trajectories naturally reside in a multi-dimensional attribute space that includes not only spatial coordinates (e.g., data center location, hop latency) but also temporal, log-

Manuscript received June 25, 2025; revised August 14, 2025. Corresponding author: Lulu Wang (email: luluwang@mail.bnu.edu.cn ).

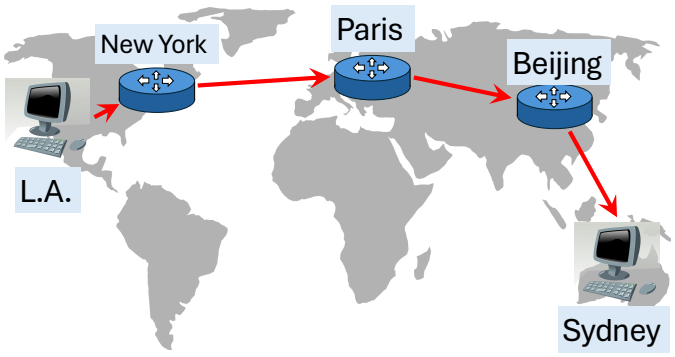


Fig. 1: Message Transmission

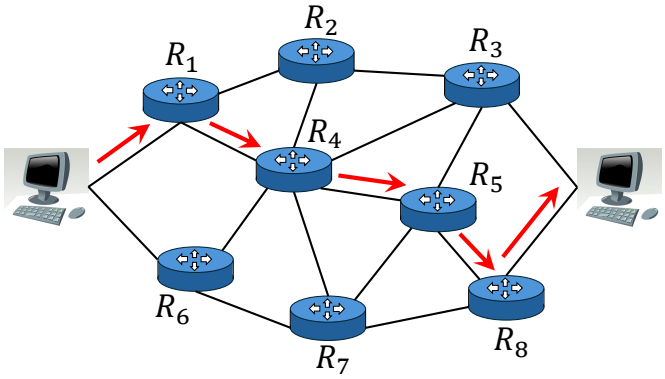


Fig. 2: Network Trajectory

ical, and application-level features (e.g., protocol type, transmission timestamp, routing cost, encryption status). The multi-dimensional trajectory cannot be projected meaningfully onto 2-dimensional planes. As a result, existing algorithms are neither accurate nor generalizable to these multi-dimensional network scenarios.

To address this, we formulate a new problem: Network Trajectory Protection (NTP). The goal is to simplify network trajectories under a Euclidean distance-based loss metric, while maintaining sequential consistency and minimizing information distortion. We define the loss between the original and simplified trajectory using a node-wise distance function, and evaluate the worst-case distortion introduced by simplification. Our loss model generalizes the classic "min-loss" formulation to arbitrary multi-dimensional spaces with sequential constraints.

Formally, given a trajectory $T$ of $n$ nodes and a positive integer $k$, we aim to find a simplified trajectory $S$ that preserves order and minimizes the maximum node-wise distance. We denote the trajectory loss by $\epsilon(S|T)$. The problem is to find $S \subset T$ such that $\min_{S \subset T, |S|=k} \epsilon(S|T)$.

We show this problem is NP-hard, which is obtained via a reduction from the classical Set Cover problem. To address the computational intractability, we propose a greedy algorithm that incrementally builds the simplified trajectory by always inserting the node with the largest *local loss*. At each step, the algorithm identifies the node in the original trajectory whose distance to its enclosing nodes in the simplified trajectory is maximal and adds it to the simplification trajectory. This process continues until the number of nodes in the simplified trajectory reaches $k$. We implement our algorithm and conduct extensive experiments on multi-dimensional trajectory datasets. Our evaluation demonstrates that the proposed algorithm performs well.

This paper makes the following contributions:

- We propose a new problem, called Network Trajectory Protection (NTP). The goal is to simplify the multi-dimensional network trajectory while preserving the essential structure.
- We formally define a Euclidean distance-based loss metric and prove that minimizing trajectory loss under a budget constraint is NP-hard.
- We design an efficient greedy approximation algorithm to solve the NTP problem.
- We empirically validate the proposed algorithm on network trajectory datasets and demonstrate its practical advantages over baseline algorithms.

The remainder of the paper is organized as follows. Section II presents the problem definition. Section III presents our proposed algorithm. Section IV demonstrates our experiment. Section V presents the related work on trajectory simplification. Finally, Section VI concludes this paper.

## II. Problem Definition

In this section, we present the problem statement. We first introduce the network trajectory with relevant notations. The trajectory loss is discussed. Then, we show the formal problem definition and prove that it is NP-hard.

### A. Network Trajectory

A network trajectory, denoted by $T = [t_1, t_2, \ldots, t_n]$, is a sequence of network nodes ordered by time. Each $t_i$, where $i \in [1, n]$, represents a $d$-dimensional feature vector of the node. Here, the dimension $d$ can be any positive integer (e.g., 5), representing attributes such as geographical location, protocol features, port behavior, timestamp signatures, etc. The subscript $i$ represents the position of the node in the trajectory according to the order. For example, as shown in Figure 2, the message transits from one terminal to another. The network nodes passed through are in the order $R_1 \rightarrow R_4 \rightarrow R_5 \rightarrow R_8$. In this case, the trajectory can be recorded as $T = [t_1 = R_1, t_2 = R_4, t_3 = R_5, t_4 = R_8]$.

A simplified trajectory $S = [s_1, s_2, \ldots, s_r]$ is a subsequence of $T$, i.e., $S \subset T$ and the order of nodes in $S$ follows the original order in $T$. Let $|\cdot|$ denote the cardinality of a trajectory, i.e., the number of nodes in the trajectory. The simplified one $S$ of a trajectory $T$ has a smaller cardinality than $T$, i.e., $|S| < |T|$. For instance, $S = [s_1 = R_1, s_2 = R_8]$ is a simplified trajectory of $T = [t_1 = R_1, t_2 = R_4, t_3 = R_5, t_4 = R_8]$ with $|S| < |T|$ since $|S| = 2$ and $|T| = 4$.

TABLE I: Concepts and Notations

| Notation | Meaning |
|---|---|
| $T$ | A $d$-dimensional trajectory in the format of $[t_1, t_2, ..., t_n]$. |
| $d$ | The number of dimensions of nodes. |
| $n$ | The number of nodes in the trajectory. |
| $t_i$ | A node with the index $i$ in the trajectory $T$. |
| $S$ | A simplified trajectory of $T$ in the form of $[t_{m_1}, t_{m_2}, \ldots, t_{m_r}]$, where $1 \leq m_1 \leq m_2 \cdots \leq m_r \leq n$. |
| $k$ | The allowed number of nodes in the simplified trajectory $S$. |
| $\lvert \cdot \rvert$ | The number of nodes in the trajectory. |
| $D(\cdot, \cdot)$ | The Euclidean distance between two nodes. |
| $L(t_i)$ | The local loss of node $t_i$. |
| $\epsilon(S\lvert T)$ | The trajectory loss of $S$ w.r.t. $T$. |

Given a trajectory $T$ and its simplified one $S$, we denote the trajectory loss of $S$ compared to $T$ by $\epsilon(S|T)$. There are many different error functions proposed to measure the loss. It is worth noting that our algorithm framework is general. It can accommodate different distance metrics, such as cosine distance and Manhattan distance, depending on the application. However, to facilitate the presentation and ensure fair comparison with existing studies, we adopt *Euclidean Distance* [6], [7], [9], [10] as the error function in this paper. The idea is as follows. For each node $t_i$ in the trajectory $T$, we can find two nodes $s_j = t_r$ and $s_{j+1} = t_{r'}$ in the simplified trajectory $S$ such that $r \leq i < r'$. Intuitively, the trajectory segment $[t_r, t_{r+1}, \ldots, t_{r'-1}]$ of $T$ includes the node $t_i$, i.e., $t_i \in [t_r, t_{r+1}, \ldots, t_{r'-1}]$. We define the local loss $L(t_i)$ of $t_i$ by the Euclidean distance between nodes $t_i$ and $s_j = t_r$, i.e.,

$$L(t_i) = D(t_i, t_r)$$

In this case, the loss $\epsilon(S|T)$ is defined to be the maximum local loss across all nodes in $T$, i.e.,

$$\epsilon(S|T) = \max_{t_i \in T} L(t_i)$$

*Definition* 1 (Trajectory Loss). Given a trajectory $T = [t_1, t_2, \ldots, t_n]$ and its simplified version $S = [s_1 = t_{m_1}, s_2 = t_{m_2}, \ldots, s_k = t_{m_r}]$, where $m_1 < m_2 < \cdots < m_r$, the trajectory error $\epsilon(S|T)$ between $T$ and $S$ is defined as $\max_{t_i \in T} \max_{t_i \in T} D(t_i, t_{m_j})$, where $t_{m_j}, t_{m_{j+1}} \in T$ and $t_{m_j} \leq t_i < t_{m_{j+1}}$.

*Example* 1. Consider a trajectory $T = [t_1, t_2, t_3, t_4]$ with $t_1 = R_1$, $t_2 = R_4$, $t_3 = R_5$, and $t_4 = R_8$, as shown in Figure 2. Suppose that the simplified version is $S = [t_1, t_4]$. The local losses for the nodes in $T$ are: $L(t_1) = D(t_1, t_1), L(t_2) = D(t_2, t_1), L(t_3) = D(t_3, t_1)$ and $L(t_4) = D(t_4, t_4)$. Since $D(t_1, t_1) = D(t_4, t_4) = 0$, the trajectory loss of $S$ compared to $T$ is $\max\{L(t_2), L(t_3)\}$.

### B. Problem NTP

Based on the definition of trajectory loss, the problem of Network Trajectory Protection, denoted by NTP for short, can be formalized. Our goal is to remove some nodes from the trajectory so that the detailed routing information can be hidden, while ensuring the simplified trajectory still represents the original trajectory with as small a trajectory loss as possible. Formally, the problem is defined as follows.

*Problem* 1 (Network Trajectory Protection (NTP)). Given a trajectory $T$ and an integer $k$, the NTP problem is to find a simplified trajectory $S$ of $T$ such that $|S| \leq k$ and the trajectory loss $\epsilon(S|T)$ is minimized.

We prove the NP-hardness of NTP by designing a decision version and deriving a reduction from the Set Cover problem [11], [12].

*Theorem* 1. The Network Trajectory Protection (NTP) problem is NP-hard.

*Proof.* Consider the decision variant of the NTP problem: given a trajectory $T$, a trajectory loss threshold $\varepsilon$, and an integer $k$, does there exist a simplified trajectory $S$ with $|S| \leq k$ such that $\epsilon(S|T) \leq \varepsilon$?

This problem asks whether a trajectory can be approximated using at most $k$ nodes such that the trajectory loss is below $\varepsilon$. Since it is a decision version of the NTP problem, if it is NP-hard, the NTP problem is also NP-hard. In the following, we prove that this decision problem is NP-hard via a reduction from the classical Set Cover problem [11], [12].

In the Set Cover problem, we are given a universe $\mathcal{U} = \{u_1, u_2, ..., u_n\}$ and a collection of subsets $\mathcal{P} = \{P_1, P_2, ..., P_m\}$, and we wish to determine whether there exists $k$ subsets whose union covers $U$. We can encode each element $u_i$ as a trajectory point $t_i \in T$. Each subset $P_j$ corresponds to two nodes $t_i, t_j \in T$, where the trajectory loss between the trajectory $[t_i, t_j]$ and the trajectory $[t_i, t_{i+1}, \ldots, t_{j-1}, t_j]$ is no greater than $\varepsilon$. We say these two nodes $t_i$ and $t_j$ covers the nodes $t_i, t_{i+1}, \ldots, t_{j-1}, t_j$. In this case, selecting $k$ subsets to cover $U$ is equivalent to selecting $k$ pairs of nodes in $T$ to cover the entire trajectory within the specified error bound.

Thus, solving the decision version of NTP would solve the Set Cover problem. The NTP problem is NP-hard. □

## III. Algorithm

Given the NP-hardness of the Network Trajectory Protection (NTP) problem, finding an optimal simplification

of a trajectory is computationally infeasible for large-scale network trajectories. Therefore, we seek efficient approximation algorithms that can strike a balance between runtime efficiency and solution quality. A natural and powerful strategy for this purpose is greedy selection.

### A. Greedy Algorithm

We propose an *inclusion-based greedy* algorithm to construct the simplified trajectory $S$. The main idea is to iteratively include nodes from the trajectory $T$ into $S$ in a way that greedily reduces the maximum loss, until a budget of $k$ nodes is reached. Since the NTP problem is NP-hard (as shown in Section II-B), this greedy manner provides a practical approximation by focusing on worst-case loss reduction in each iteration.

Initially, the simplification trajectory $S$ contains only the two end nodes of the trajectory $T$, i.e., $S = [t_1, t_n]$. In each iteration, one node is selected from the trajectory $T$ and added to $S$. If there are $k$ nodes in $S$, i.e., $|S| \geq k$, the process stops. Otherwise, another iteration starts.

During the process, the core part is selecting a node from the trajectory $T$ to add it to $S$. We utilize a greedy idea: *among all nodes in the trajectory $T$, those nodes that deviate most from the current simplified trajectory should be prioritized for inclusion*. This is because these nodes represent the parts where the current simplification performs the worst. By adding them to the simplified trajectory $S$ iteratively, we continuously reduce the worst-case distortion in a cost-effective manner.

Specifically, in each iteration, for each node $t_i \in T$ that are not yet included in the simplified trajectory $S$, we find the pair of adjacent nodes $s_{m_j}, s_{m_{j+1}} \in S$ such that $m_j \leq i < m_{j+1}$. This means that $t_i$ lies between these two nodes. We then calculate (1) the Euclidean distance $D(t_i, t_{m_j})$ between $t_i$ and $t_{m_j}$ and (2) the Euclidean distance $D(t_i, t_{m_{j+1}})$ between $t_i$ and $t_{m_{j+1}}$. The local loss $L(t_i)$ of $t_i$ is the minimum Euclidean distance of the two, i.e., $L(t_i) = \min\{D(t_i, t_{m_j}), D(t_i, t_{m_{j+1}})\}$. This loss reflects how poorly $t_i$ is represented by the simplified trajectory. After computing all local losses, the node with the highest local loss is selected and added to the simplified trajectory $S$. This step ensures that the largest deviation is greedily reduced at each iteration.

*Example* 2. Consider Figure 3 with a trajectory $T = [t_1, t_2, t_3, t_4]$. Suppose that the current simplified version is $S = [t_1, t_4]$. There are two nodes in $T$ but not in $S$: $t_2$ and $t_3$. We calculate the local losses of these two nodes $L(t_2) = \min\{D(t_2, t_1), D(t_2, t_4)\}$ and $L(t_3) = \min\{D(t_3, t_1), D(t_3, t_4)\}$, respectively. If $L(t_2) > L(t_3)$, node $t_2$ is selected and added to the simplified trajectory $S$.

### B. Summary and Analysis

We summarize our algorithm with the pseudocode shown in Algorithm 1. The input to the algorithm consists of a network trajectory $T = [t_1, t_2, \ldots, t_n]$ and a positive integer $k$, where $2 \leq k < n$. This integer $k$ specifies the desired number of nodes in the simplified trajectory $S$. Initially, the simplified trajectory $S$ is set to contain the two end nodes $t_1$ and $t_n$, i.e., $S = [t_1, t_n]$ (line 1). In each iteration, let the current simplified trajectory be $S = [t_{m_1}, t_{m_2}, \ldots, t_{m_r}]$, where each $m_j$ is an index such that $1 = m_1 < m_2 < \cdots < m_r = n$. For each adjacent pair $t_{m_j}$ and $t_{m_{j+1}}$ in $S$ (line 4), let us consider the sub-sequence $[t_{m_j}, t_{m_j+1}, t_{m_j+2}, \ldots t_{m_{j+1}}] \subset T$ (line 5). For each node $t_i$ in this sub-sequence, we compute its local loss $L(t_i) = \min\{D(t_i, t_{m_j}), D(t_i, t_{m_{j+1}})\}$ (line 6), and thus, identify the node $t^*$ in this sub-sequence that has the maximum local loss (lines 7-10). After going through all adjacent pairs $t_{m_j}$ and $t_{m_{j+1}}$ in $S$, we can find the node $t^*$ in $T$ with the maximal local loss. We insert $t^*$ into the simplified trajectory $S$ at the appropriate location (line 14). The process iterates until the simplified trajectory $S$ contains $k$ nodes, i.e., $|S| = k$ (line 2), and the simplified trajectory is returned.

*Example* 3. We now illustrate the algorithm with a concrete example. Consider a trajectory $T = [t_1, t_2, t_3, t_4, t_5, t_6]$, where each node is a 2-dimensional vector as shown in Figure 4. Our goal is to simplify this

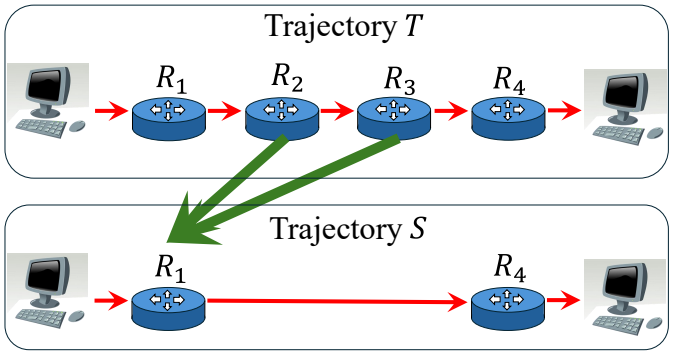


Fig. 3: Greedy Algorithm

**Algorithm 1:** Inclusion-Based Greedy Algorithm

**Require:** Trajectory $T = [t_1, t_2, \ldots, t_n]$, parameter $k$
**Ensure:** Simplified trajectory $S$ with $|S| \leq k$
1: Initialize $S \leftarrow [t_1, t_n]$
2: **while** $|S| < k$ **do**
3: $t^* \leftarrow$ None, $L_{\max} \leftarrow -\infty$
4: **for all** $j = 1$ to $|S| - 1$ **do**
5: **for all** $t_i \in T$ such that $m_j < i < m_{j+1}$ **do**
6: $L(t_i) = \min\{D(t_i, t_{m_j}), D(t_i, t_{m_{j+1}})\}$
7: **if** $L(t_i) > L_{\max}$ **then**
8: $L_{\max} \leftarrow L(t_i)$
9: $t^* \leftarrow t_i$
10: $j^* \leftarrow j + 1$
11: **end if**
12: **end for**
13: **end for**
14: Insert $t^*$ into $S$ at position $j^*$
15: **end while**
16: **return** $S$

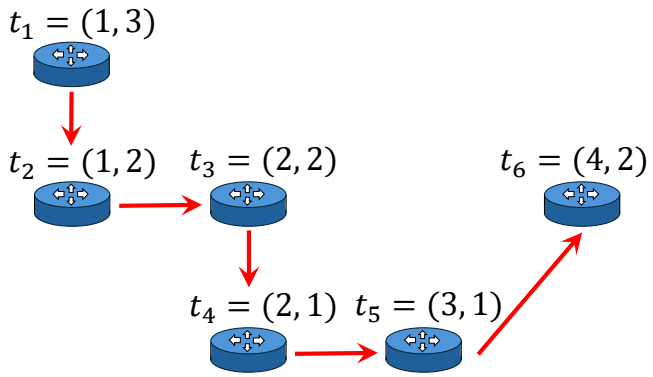


Fig. 4: Trajectory Nodes

trajectory by reducing its number of nodes to $k = 4$. Initially, the simplified trajectory is set to $S = [t_1.t_6]$. In the first iteration, there is only one pair of adjacent nodes $t_1$ and $t_6$ in $S$. We find the nodes $t_i$ such that $1 < i < 6$ and compute their local loss $L(t_i)$, respectively.

$$L(t_2) = \min\{D(t_2, t_1), D(t_2, t_6)\} = 1.00$$

$$L(t_3) = \min\{D(t_3, t_1), D(t_3, t_6)\} = 1.41$$

$$L(t_4) = \min\{D(t_4, t_1), D(t_4, t_6)\} = 2.23$$

$$L(t_5) = \min\{D(t_5, t_1), D(t_5, t_6)\} = 1.41$$

Since the maximum local loss occurs at node $t_4$, we insert $t_4$ into $S$, and thus, $S = [t_1, t_4, t_6]$.

In the second iteration, there are two pairs of adjacent nodes in $S$. Consider the first pair $t_1$ and $t_4$. We find the nodes $t_i$ such that $1 < i < 4$ and compute their local loss, respectively.

$$L(t_2) = \min\{D(t_2, t_1), D(t_2, t_4)\} = 1.00$$

$$L(t_3) = \min\{D(t_3, t_1), D(t_3, t_4)\} = 1.00$$

Consider the second pair $t_4$ and $t_6$. We find the nodes $t_i$ such that $4 < i < 6$ and compute their local loss $L(t_i)$, respectively.

$$L(t_5) = \min\{D(t_5, t_4), D(t_5, t_6)\} = 1.00$$

Since the nodes have the same local loss, we break the tie by randomly selecting one. If node $t_2$ is selected, we insert $t_2$ into $S$, and thus, $S = [t_1, t_2, t_4, t_6]$. Since $|S| \geq 4 = k$, the simplified trajectory $S = [t_1, t_2, t_4, t_6]$ is returned.

Now, we provide the theoretical analysis of our algorithm. Theorem 2 shows the time complexity.

*Theorem* 2. Given a trajectory $T = [t_1, t_2, \ldots, t_n]$ and a positive integer $k$ which is the desired size of the simplified trajectory, our algorithm runs in $O(kn)$ time.

*Proof.* Let us examine the operations performed during the course of the algorithm.

The simplified trajectory $S$ is initialized with two end nodes, i.e., $S = [t_1, t_n]$. It takes constant time.

In each iteration, the algorithm considers each pair of adjacent nodes $t_{m_j}$ and $t_{m_{j+1}}$ in $S$. For each pair, it examines all nodes in $T$ that lie between them (i.e., $t_i \in T$ such that $m_j < i < m_{j+1}$). Thus, each node in $T$ is examined at most once per iteration. Since there are $n$ nodes in $T$, each iteration involves at most $O(n)$ work to compute the local losses and select the node with the maximum deviation.

Since exactly one node is added to $S$ per iteration, after $k - 2$ iterations, there are $k$ nodes in $S$ (including the two end nodes). Therefore, the algorithm terminates after $k - 2$ iterations. In this case, the total runtime is $O((k - 2)n) = O(kn)$. □

## IV. Experiments

This section presents our empirical study on the network trajectory protection. We present the setting of our experiments in Section IV-A. Section IV-A demonstrates the results, and a summary is shown in Section IV-C.

### A. Experimental Setting

**Evaluation Platform.** The implementation of all algorithms is carried out in Python. All the experiments are conducted on a Mac machine equipped with an M3 chip and 128.0GB RAM.

**Datasets.** Our experimental evaluations are conducted on six trajectory datasets, namely Traj-4D, Traj-5D, Traj-6D, Random-4D, Random-5D, and Random-6D. The first three datasets are generated by applying the ActiveRanking algorithm [13] [14], [15] with 4, 5, and 6 attributes, respectively. This algorithm is commonly used in many systems. During the process, multiple attributes are considered, each attribute representing a distinct dimension. As a result, each state during the algorithm processing can be represented as a node (i.e., a concatenation of these attributes), and the entire sequence can be modeled as a trajectory. The last three datasets are generated randomly. The information about the trajectory datasets is detailed in Table II.

**Baseline.** Most existing trajectory simplification algorithms are designed specifically for 2-dimensional settings and cannot be directly applied to high-dimensional trajectories. To enable comparison in multi-dimensional scenarios, we implement a simple baseline based on random selection. Specifically, the baseline iteratively selects $k$ nodes at random from the original trajectory to construct the simplified version.

**Parameter Settings and Measurements.** Let $n$ denote the length of the original trajectory. We vary the simplification budget $k$ (i.e., the number of nodes allowed in the simplified trajectory) according to a set of predefined ratios: $k = ratio \times n$, where $ratio \in \{0.1, 0.2, 0.3, 0.4, 0.5\}$. We evaluate each method in terms of two metrics: (1) the trajectory loss, measured as the maximum node-wise Euclidean distance between the original and simplified trajectories; and (2) the execution time in seconds. For

TABLE II: Statistics of Trajectory Datasets

| Dataset | Dimensionality | No. of Trajectories | Reference |
|---|---|---|---|
| Traj-4D | 4 | 2,599 | [13] |
| Traj-5D | 5 | 2,199 | [13] |
| Traj-6D | 6 | 2,449 | [13] |
| Random-4D | 4 | 3,000 | - |
| Random-5D | 5 | 3,000 | - |
| Random-6D | 6 | 3,000 | - |

each configuration, we randomly sample 100 trajectories from the dataset and report the average results to ensure statistical stability.

### B. Experimental Results

We evaluate the effectiveness and efficiency of our proposed greedy algorithm by comparing it against a baseline method, referred to as RANDOM.

Figures 5, 6, and 7 present the results on three datasets: Traj-4D, Traj-5D, and Traj-6D, respectively. Across these datasets, our greedy algorithm consistently achieves significantly lower trajectory loss than the RANDOM baseline. In Figure 5, the greedy method maintains a trajectory loss below 1.0 in most cases, while the loss from RANDOM can be up to 1.2 times higher. This performance gap becomes even more pronounced in Figure 6, particularly under lower simplification ratios, highlighting the greedy algorithm's advantage when fewer nodes are retained. Similarly, Figure 7 shows that our method remains robust as dimensionality increases, consistently outperforming the baseline in all tested scenarios. In addition, the trajectory loss achieved by our algorithm decreases as the ratio increases. This trend is expected, as an increasing ratio leads to a larger $k$. There are more nodes to be retained, thereby enabling a closer approximation to the original trajectory.

To further evaluate the robustness of our proposed algorithm under different data distributions, we extended our experiments to the randomly generated datasets: Random-4D, Random-5D, and Random-6D. Figures 8, 9, and 10 present the results on three datasets: Random-4D, Random-5D, and Random-6D, respectively. Similar to the observations on the trajectory-based datasets, our greedy algorithm consistently outperforms the RANDOM baseline in terms of trajectory loss across all dimensions. Notably, this advantage remains consistent regardless of the dimensionality increase from 4D to 6D, validating the algorithm's scalability. This demonstrates that the algorithm can effectively capture essential structural information even when dealing with multi-dimensional data that lacks the inherent sequential patterns of real-world trajectories.

In terms of runtime, the greedy algorithm remains efficient across all settings. Although it is naturally slower than the lightweight RANDOM baseline due to its optimization steps, the overall execution time remains

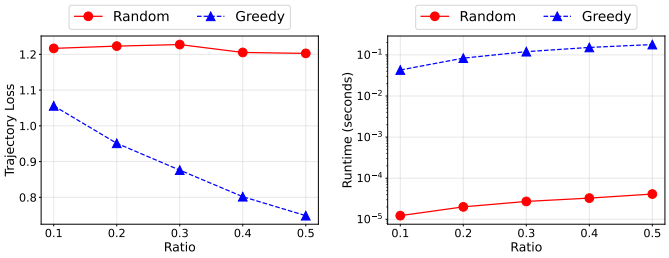


Fig. 5: Traj-4D

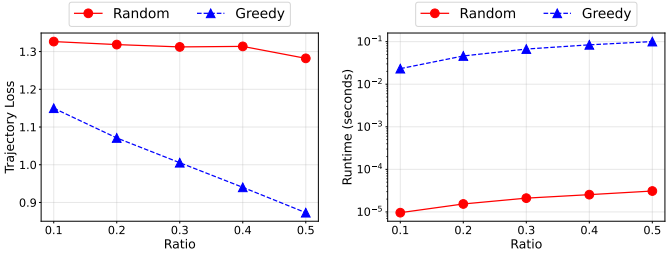


Fig. 6: Traj-5D

low. For instance, in Figure 5, the greedy algorithm consistently completes within one second across all ratios. These results demonstrate that our greedy method is both effective and practical for multi-dimensional trajectory simplification tasks.

### C. Summary

The experimental results demonstrate the practical effectiveness of our proposed greedy algorithm for multi-dimensional trajectory simplification. Across all tested datasets, including both the datasets generated by the ACTIVERANKING algorithm (Traj-4D, Traj-5D, Traj-6D) and the randomly generated datasets (Random-4D, Random-5D, Random-6D), our algorithm consistently achieves lower trajectory loss compared to the RANDOM baseline. In some configurations, the greedy algorithm yields up to a 30% improvement, highlighting its ability to retain representative nodes that preserve the essential structure of the original trajectory. Furthermore, the algorithm exhibits strong scalability. As the dimensionality of the nodes increases from 4 to 6, the runtime of the greedy algorithm grows only moderately, remaining within a practical range for real-world applications. This performance makes it suitable for online or near-real-time sce-

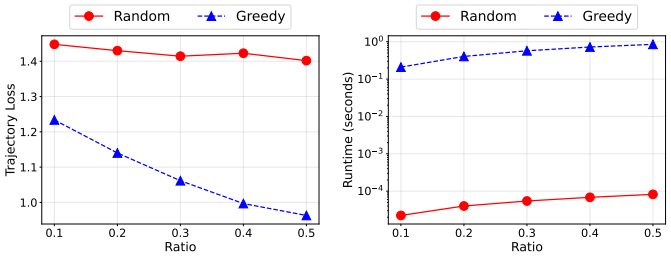


Fig. 7: Traj-6D

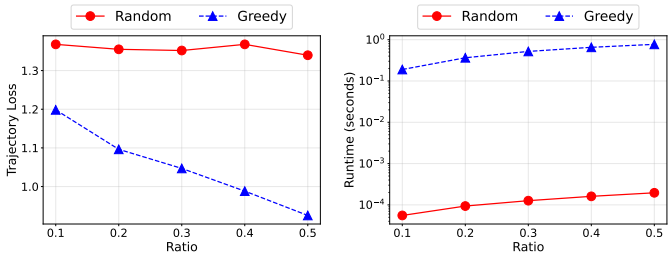


Fig. 9: Random-5D

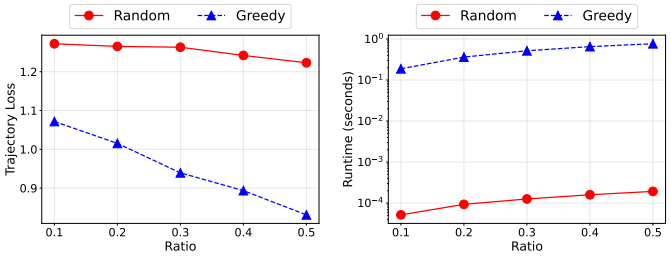


Fig. 8: Random-4D

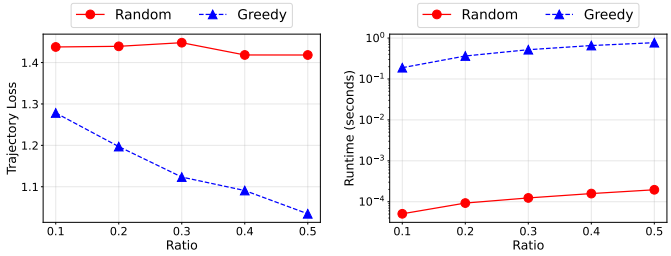


Fig. 10: Random-6D

narios. These findings support the use of our algorithms in the network trajectory protection application.

## V. Related Work

In this section, we discuss several 2-dimensional trajectory simplification algorithms. We classify them into two types. The first type includes the rule-based algorithms, while the second type considers the reinforcement learning-based algorithms.

### A. Rule-Based

A substantial body of prior work has attempted to address the *Min-Error* trajectory simplification problem, with several studies proposing heuristic or rule-based methods tailored to specific error metrics [7]–[9], [16]. One of the earliest solutions by Bellman [17] employs dynamic programming. It achieves optimal results but suffers from a prohibitive computational cost of at least $O(n^3)$ for a trajectory with $n$ nodes. This high cost has driven the research community to explore more scalable alternatives.

Consequently, numerous follow-up studies [6]–[9], [16], [18], [19] have proposed greedy approximation algorithms designed to reduce computational overhead. These algorithms typically fall into two categories: some iteratively remove non-essential nodes from the original trajectory, while others incrementally build a simplified trajectory starting with the end nodes. It is important to note that these greedy strategies are usually tightly coupled to the specific error function for which they were designed and may not perform well outside those predefined contexts.

A broader overview and empirical evaluation of these techniques can be found in the survey by Zheng et al. [20], which systematically compares many representative algorithms.

In parallel, significant progress has also been made on the *Error-Bounded* trajectory simplification problem, where the objective is to reduce the number of nodes while ensuring that the simplification error remains within a given threshold. Prominent algorithms in this line of work include [6], [21]–[27]. Most of these techniques adopt a standard three-phase sliding window framework: (1) initialize a window over two adjacent nodes, (2) expand the window by including more nodes until the error exceeds the threshold, and (3) output the last valid node and reinitialize the window. While their window expansion policies may differ, the overarching structure remains largely consistent.

These algorithms are predominantly rule-based, relying on heuristics tailored to specific error formulations, and are often restricted to two-dimensional spatial data. As a result, their applicability is limited when dealing with new or complex error metrics, especially in higher-dimensional settings or emerging application domains where standard error models are insufficient.

### B. Reinforcement Learning-Based

A distinct line of research has explored framing trajectory simplification as a reinforcement learning (RL) problem. In the RL framework, an agent interacts with an environment in order to maximize the cumulative reward over time. The environment is formally characterized by a set of components: states, actions, rewards, state transitions, and a discount factor. The state captures the current status of the agent, while the action space defines the available decisions at each state. The transition function determines how the environment evolves when an action is taken. Each state-action pair is associated

with an immediate reward, and the reward accumulated at step $t$ is typically weighted by a discount factor $\gamma^t$, where $\gamma \in [0, 1]$ controls the importance of future rewards. The overall goal is to find a policy—a mapping from states to actions—that maximizes the sum of all discounted rewards. Under a fixed policy, the RL process can be formalized as a Markov Decision Process (MDP).

One of the earliest efforts to apply RL to trajectory simplification was introduced in [28], which targets the Min-Error trajectory simplification problem in 2-dimensional space. The authors recast the simplification process as an RL problem, where at each step the agent chooses to either retain or discard the current trajectory node. The reward signal is defined as the reduction in simplification error caused by the action. The state representation encodes a set of handcrafted features derived from the local geometry and the partially simplified trajectory observed so far. The transition model updates these features based on the selected action.

Building on this idea, [29] proposed a multi-agent deep reinforcement learning approach to address the Error-Bounded version of the problem. In their design, one agent is responsible for determining whether a node should be kept, thereby initiating a new simplification window, while a second agent identifies redundant nodes within that window that can be safely excluded without violating the predefined error constraint. Both agents rely on explicitly engineered state representations, action definitions, and reward functions that are tightly coupled to the 2-dimensional spatial structure.

Later, [30] extended the RL-based paradigm to operate over an entire trajectory dataset, rather than individual trajectories, aiming to produce collective simplification policies that generalize across samples. Separately, [31] introduced a novel sequence-to-sequence-to-sequence (S3) neural framework for 2-dimensional trajectory simplification. Their model incorporates a trainable encoder-decoder-based "compressor" and a learnable "reconstructor" that estimates the reconstruction error between the simplified and original trajectories, enabling fully differentiable end-to-end optimization.

Despite these advances, a common limitation remains: these approaches are inherently restricted to 2-dimensional trajectories. The structure of their reward models, state encodings, and even action definitions heavily rely on 2-dimensional geometric priors (e.g., angles, distances in a plane), making them difficult or infeasible to generalize to high-dimensional trajectory data, such as those found in network systems, robotics, or multivariate temporal domains.

## VI. Conclusion

This work presents a new problem, Network Trajectory Protection (NTP), motivated by the increasing need to balance trajectory utility and privacy in multi-dimensional network environments. While traditional trajectory simplification algorithms are confined to 2-dimensional settings, our formulation generalizes to arbitrary dimensions. We formally prove that the NTP problem is NP-hard, highlighting its theoretical complexity. To address this challenge, we design a greedy algorithm. The experimental results across various datasets show the superiority of the proposed algorithm. Future work may extend this direction to dynamic networks.

## References


[1] S. Chakrabarty and D. W. Engels, "Secure smart cities framework using iot and ai," in *2020 IEEE Global Conference on Artificial Intelligence and Internet of Things (GCAIoT)*, 2020.

[2] X. Zhao, J. Qiao, X. Huang, C. Wang, S. Song, and J. Wang, "Apache tsfile: An iot-native time series file format," *Proceedings of the VLDB Endowment*, vol. 17, no. 12, pp. 4064–4076, 2024.

[3] T. Chang and O. Khan, "Rapdad: A low latency desynchronization approach for 6tisch-based asset tracking networks," *IEEE Transactions on Industrial Informatics*, pp. 1–13, 2025.

[4] A. Rejeb, K. Rejeb, H. Treiblmaier, A. Appolloni, S. Alghamdi, Y. Alhasawi, and M. Iranmanesh, "The internet of things (iot) in healthcare: Taking stock and moving forward," *Internet of Things*, vol. 22, p. 100721, 2023.

[5] F. Al-Turjman, M. H. Nawaz, and U. D. Ulusar, "Intelligence in the internet of medical things era: A systematic review of current and future trends," *Computer Communications*, vol. 150, pp. 644–660, 2020.

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

[7] M. Potamias, K. Patroumpas, and T. Sellis, "Sampling trajectory streams with spatiotemporal criteria," in *18th International Conference on Scientific and Statistical Database Management*, 2006.

[8] H. Li, L. Kulik, and K. Ramamohanarao, "Spatio-temporal trajectory simplification for inferring travel paths," in *SIGSPATIAL*, 2014, pp. 63–72.

[9] 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*, 2011, pp. 1–8.

[10] 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*, vol. 18, no. 3, pp. 435–460, 2014.

[11] V. Chvatal, "A greedy heuristic for the set-covering problem," *Math. Oper. Res.*, vol. 4, no. 3, p. 233–235, Aug. 1979.

[12] M. R. Garey and D. S. Johnson, *Computers and Intractability; A Guide to the Theory of NP-Completeness*. USA: W. H. Freeman & Co., 1990.

[13] K. G. Jamieson and R. D. Nowak, "Active ranking using pairwise comparisons," in *NIPS*, 2011.

[14] W. Wang, R. C.-W. Wong, and M. Xie, "Interactive search for one of the top-k," in *SIGMOD*. New York, NY, USA: ACM, 2021.

[15] R. C.-W. W. Weicheng Wang and M. Xie, "Interactive search with mixed attributes," in *ICDE*, 2023.

[16] 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*, vol. 18, no. 3, pp. 435–460, 2014.

[17] R. Bellman, "On the approximation of curves by line segments using dynamic programming," *Communications of the ACM*, vol. 4, no. 6, p. 284, 1961.

[18] 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*. IEEE, 2001, pp. 289–296.

[19] C. Long, R. C.-W. Wong, and H. Jagadish, "Trajectory simplification: On minimizing the direction-based error," *VLDB*, vol. 8, no. 1, pp. 49–60, 2014.

[20] D. Zhang, M. Ding, D. Yang, Y. Liu, J. Fan, and H. T. Shen, "Trajectory simplification: an experimental study and quality analysis," *VLDB*, 2018.

[21] X. Lin, J. Jiang, S. Ma, Y. Zuo, and C. Hu, "One-pass trajectory simplification using the synchronous euclidean distance," *The VLDB Journal*, vol. 28, no. 6, pp. 897–921, 2019.

[22] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, and R. Jurdak, "Bounded quadrant system: Error-bounded trajectory compression on the go," in *ICDE*, 2015, pp. 987–998.

[23] 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*, vol. 28, no. 11, pp. 2827–2841, 2016.
[24] X. L. S. Ma and H. Z. T. W. J. Huai, "One-pass error bounded trajectory simplification," *VLDB*, vol. 10, no. 7, 2017.
[25] C. Long, R. C.-W. Wong, and H. Jagadish, "Direction-preserving trajectory simplification," *VLDB*, vol. 6, no. 10, pp. 949–960, 2013.
[26] 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.
[27] W. Cao and Y. Li, "Dots: An online and near-optimal trajectory simplification algorithm," *Journal of Systems and Software*, 2017.
[28] Z. Wang, C. Long, and G. Cong, "Trajectory simplification with reinforcement learning," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 2021, pp. 684–695.
[29] 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*, 2021, pp. 1758–1768.
[30] Z. Wang, C. Long, G. Cong, and C. S. Jensen, "Collectively simplifying trajectories in a database: A query accuracy driven approach," in *ICDE*, 2024, pp. 4383–4395.
[31] Z. Fang, C. He, L. Chen, D. Hu, Q. Sun, L. Li, and Y. Gao, "A lightweight framework for fast trajectory simplification," in *ICDE*, 2023, pp. 2386–2399.

**Qianting Chen** is currently a Research Assistant at Shenzhen University, China. She obtained a B.S. degree in Computer Science from Shenzhen University in 2017 and is currently pursuing a M.S. degree in Artificial Intelligence Application at the University of Macau. Her current research interests include database systems, data mining, deep learning.

**Lulu Wang** received the M.Sc. degree in Systems Science from Beijing University of Posts and Telecommunications in 2023. She is currently pursuing the Ph.D. degree in Computer Science at Beijing Normal University. Her research interests include database systems and causal inference.

**Ke Chao** received the BS degree in Computer Science and Technology from Beijing Normal University in 2022. She is currently working toward the Ph.D. degree in Computer Science at Beijing Normal University. Her research interests include crowdsourcing, causal inference and game theory.

**Weicheng Wang** is currently a Post-Doctoral Fellow at the Chinese University of Hong Kong. He worked as a Post-Doctoral Fellow at the Hong Kong University of Science and Technology in 2024. He received his Ph.D. degree from the Hong Kong University of Science and Technology in 2023 and his BEng degree from the Beijing Normal University in 2018. His research interests include database, data mining, and machine learning.

**Rui Mao** is currently a Distinguished Professor at Shenzhen University, China. He received his Ph.D. in Computer Science from the University of Texas at Austin in 2007, and his M.S. and B.Eng. degrees from the University of Science and Technology of China in 2000 and 1997, respectively. His research interests span big data analysis and management, content-based similarity search for multimedia and biological data, and data mining.