CF-TS: A General Coarse-to-Fine Method for Trajectory Simplification

Jiayi Li, Junhua Fang(✉), Zhicheng Pan, Pingfu Chao, Jiajie Xu, and Pengpeng Zhao

Department of Computer Science and Technology, Soochow University, Suzhou, China
20235227131@stu.suda.edu.cn, jhfang@suda.edu.cn, zcpan@stu.ecnu.edu.cn, pfchao@suda.edu.cn, xujj@suda.edu.cn, ppzhao@suda.edu.cn

Abstract. Trajectory simplification aims to reduce the sizes of trajectories while preserving as much information as possible, thus improving storage, processing, and transmission efficiency. Despite substantial progress of existing studies, three major limitations remain: 1) inefficient initialization, 2) rigid pre-defined thresholds, and 3) inadequate consideration of trajectory attributes. To address these challenges, we propose a novel Coarse-to-Fine framework for Trajectory Simplification, termed CF-TS, consisting of two stages. In the coarse-grained stage, CF-TS employs a customized variant of Douglas-Peucker as a robust warm-start in a lightweight way. In the fine-grained stage, Monte Carlo Tree Search iteratively refines the trajectory in regions requiring optimization based on a candidate set, which leverages a broader spectrum of features, including smoothness and direction error, to ensure a more accurate representation of original trajectory. Extensive experiments on real-world datasets demonstrate that CF-TS outperforms state-of-the-art methods in both effectiveness and efficiency, establishing itself as a novel paradigm.

Keywords: Trajectory data · Trajectory simplification · Monte Carlo Tree Search.

1 Introduction

The proliferation of GPS-enabled devices has generated extensive trajectory data that benefits a broad range of applications [10, 11]. However, managing such large-scale data presents two main challenges [13, 30]: (1) substantial storage requirements, and (2) computationally intensive analysis, e.g., querying or clustering. Fortunately, Trajectory Simplification (TS) has emerged as a promising solution [5, 8, 9, 20, 28], which involves the strategic removal of certain points from raw trajectory while preserving the essential characteristics and utility of original data. TS is now the most common lossy compression method that offers a high compression ratio with acceptable information loss.

Existing trajectory simplification methodologies can be broadly classified into two principal categories: ❶ Error-bounded, which prioritizes maximizing compression ratio while ensuring that the simplification error remains within a predefined threshold [13]. ❷ Error-driven, which seeks to identify a sub-trajectory

comprising a fixed number of points (i.e., a given "budget") to minimize the overall trajectory simplification error [30]. Our study falls within the latter category, as it allows precise control over the compression ratio and enhanced reliability of trajectory simplification by incrementally selecting the most significant points, which aligns well with practical application needs [13, 30].

Motivations. The landscape of error-driven TS is rich with proposals, encompassing both heuristic-based and learning-based approaches. For heuristic-based methods [3, 6, 9, 20], Douglas-Peucker (DP) [6] and Top-Down Time-Ratio (TD-TR) [20] are both widely recognized, which commonly employ different metrics to iteratively assess the point significance. However, they heavily rely on human-crafted rules to guide the process, limiting their adaptability. In contrast, learning-based approaches [7, 26, 28] have emerged as a promising alternative. For example, [26] models trajectory simplification as a Markov Decision Process (MDP), leveraging reinforcement learning to determine point selection collaboratively. Despite these advancements, existing methods still have yet to fully address the following limitations, which we aim to tackle with our proposal:

Naive Initialization (L1). The majority of existing algorithms initiate the simplification process by connecting the start and end points of the trajectory to form an initial baseline [26, 28]. While straightforward in concept, this strategy is computationally inefficient, as it requires numerous iterations to satisfy the storage budget, significantly increasing runtime. This inefficiency not only hinders scalability but also highlights the need for more intelligent initialization strategies to streamline the simplification process.

Rigid Error Threshold Constraints (L2). Current methods typically employ fixed error thresholds and apply the same criterion across all trajectory segments. This rigidity often results in suboptimal outcomes. Moreover, determining an appropriate threshold can be challenging for users across various scenarios [25], thereby limiting the practical applicability of these algorithms. Also, these hand-crafted rules lack robust theoretical justification, casting doubt on the guaranteed effectiveness of algorithms that rely on them [28].

Inadequate Feature Integration (L3). Previous studies [3, 6, 20, 28] primarily emphasize distance-based metrics, which are effective for preserving the geometric structure of trajectories. However, this singular focus on distance often overlooks other vital aspects of the trajectory data, such as directionality, which are equally essential for capturing the behavioral patterns of moving objects.

Solution. To address the above limitations **(L1∼L3)**, we develop a two-stage framework, named CF-TS, to perform trajectory simplification on massive data. **(1) The coarse-grained stage,** which commences with the original trajectory as input and employs a variant of the vanilla DP algorithm to swiftly reduce the data size while retaining relatively important points. This pre-simplification trajectory serves as a robust initialization for the fine-grained stage, avoiding the inefficiency caused by naive initialization (**L1**) and reducing the computational overhead of subsequent steps. Unlike traditional methods, this stage avoids reliance on rigid thresholds, as its primary role is only to provide an adaptive, heuristic-driven starting point, addressing the limitation of fixed error

constraints (**L2**). **(2) The fine-grained stage,** which applies Monte Carlo Tree Search (MCTS) to refine the pre-simplified trajectory from the coarse stage. To reduce the search space of the MCTS exploration, we construct a candidate point explorer that incorporates a broader spectrum of features, including smoothness and direction error, thereby filtering out less critical points and focusing on those that are most pertinent to the trajectory's characteristics. This strategy effectively addresses the issue of inadequate feature integration (**L3**) and improves efficiency. In summary, our contributions are as follows:

- We propose a novel coarse-to-fine trajectory simplification framework (CF-TS), which effectively balances the simplification quality and efficiency.
- We develop a variant of Douglas-Peucker algorithm, serving as a warm-start strategy for CF-TS, which significantly accelerates the simplification process.
- We design a fine-grained point complementation algorithm based on MCTS to maximize the quality of CF-TS by intelligently identifying key points.
- Extensive experiments on real-world datasets demonstrate that CF-TS outperforms state-of-the-art methods, establishing itself as a robust and efficient paradigm for trajectory simplification.

2 Related Work

Error-driven Trajectory Simplification. Extensive research has been conducted in this domain and we review the studies on the offline mode, i.e., all trajectory points are available before simplification, which is the focus of this paper. The classic Douglas-Peucker (DP) [6] algorithm adopts a top-down approach, iteratively partitioning a trajectory by comparing points to a predefined distance threshold with Perpendicular Euclidean Distance (PED) to assess simplification error. To improve computational efficiency, DP-Hull [9] leverages convex hull properties, reducing the complexity to $O(n \log n)$ while preserving the same set of points as DP. Another variant, TD-TR [20], exploits the temporal dimension with a novel metric Synchronized Euclidean Distance (SED). Different from the above three position-based simplification algorithms, Long et al. [17] introduce a direction-preserving solution, which bounds directional information loss using a graph-based approach. While these methods are effective, they require manually preset error thresholds and rely heavily on heuristic rules, limiting adaptability. More recently, AI-based solutions have gradually emerged for their promising potential in TS problems. [7] proposes a lightweight framework that integrates a Compressor and Constructor to minimize reconstruction loss, offering high efficiency but at the cost of increased simplification error. Wang et al. [26] introduce an Octree-based method with intelligent agents to optimize trajectory queries post-simplification. However, these approaches prioritize downstream usability over simplification effectiveness.

Reinforcement Learning. Reinforcement learning (RL), a machine learning technique, guides agents in making decisions to maximize cumulative rewards within an environment [24], typically modeled as a Markov Decision Process

(MDP), involving states, actions, and rewards [12]. Recently, RL has been applied successfully in trajectory analysis including trajectory simplification [22, 25], similarity search [27], and index learning [29], etc. For instance, [28] proposes the MARL4TS algorithm, which introduces two collaborative agents to perform TS in an online environment, one to decide on the window expansion and the other for determining when to open a new window.

In this paper, we apply reinforcement learning for error-driven trajectory simplification problems and distinguish our approach from the existing RL-based TS studies in three aspects. 1) A warm-start initialization strategy improves search efficiency and overall performance. 2) Enhanced feature integration guides the RL process. 3) It leverages Monte-Carlo tree search [2, 23] for the decision making and the corresponding designs (e.g., those of MDPs) are different.

3 Preliminaries

3.1 Basic Concepts

Definition 1. ***(Trajectory).*** *A trajectory* $T = \{p_1, p_2, ..., p_n\}$ *is a time-ordered sequence of GPS points, where* p_i *has the form of a triplet* (x_i, y_i, t_i)*, meaning that a moving object is located at the geographic coordinates* (x_i, y_i) *at time* t_i*.*

Definition 2. ***(Anchor Segment).*** *An anchor segment* $\overrightarrow{p_{u_i} p_{u_i+1}}$ *refers to a directed line-segment from* p_{u_i} *to* p_{u_i+1}*, which approximates those points located within the interval* (p_{u_i}, p_{u_i+1}) *in* T *during simplification.*

Definition 3. ***(Trajectory Simplification).*** *A simplified trajectory of* T*, denoted by* T'*, has the form of* $\langle p_{s_1}, p_{s_2}, ..., p_{s_m} \rangle$ *where* $m \leq n$ *and* $1 = s_1 < s_2 < ... < s_m = n$*,* n *is the number of points in* T*.*

To measure the information loss of the simplification, several error metrics have been proposed, including Perpendicular Euclidean Distance (PED) [15, 16], Synchronized Euclidean Distance (SED) [21, 22] and Direction-aware Distance (DAD) [17, 18]. Assume that p_m is located within the interval (p_s, p_e) and is to be discarded, $\overrightarrow{p_s p_e}$ is an anchor segment (Definition 2) of which. The error of a segment $\epsilon(\overrightarrow{p_s p_e})$ is defined as the maximum error a point that takes the segment as its anchor segment, denoted as:

$$\epsilon(\overrightarrow{p_s p_e}) = \max_{s \leq i < e} \epsilon(\overrightarrow{p_s p_e} | p_i), \tag{1}$$

where $\epsilon(\overrightarrow{p_s p_e})$ can be instantiated with PED, SED or DAD, and detailed definitions can be found in the evaluation paper [30]. Assume that N_s refers to the number of anchor segments of T', then the overall simplification loss $\epsilon(T')$ is defined as the average error of all its segments based on Equation (1) as follows:

$$\epsilon(T') = \frac{1}{N_s} \sum_{1 \leq s, e \leq N_s} \epsilon(\overrightarrow{p_s p_e}). \tag{2}$$

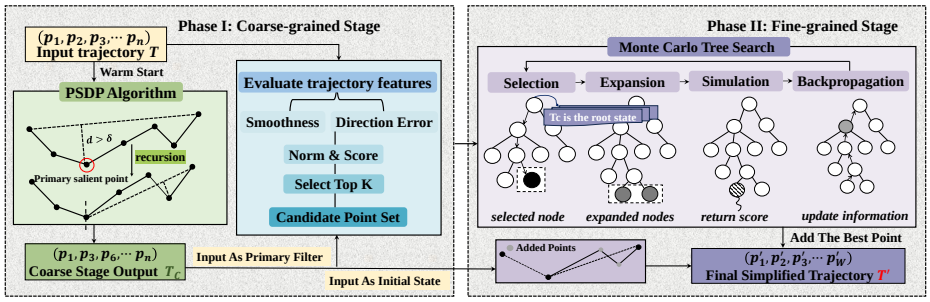


Fig. 1. The framework of CF-TS.

3.2 Problem Statement

Based on the above concepts, the error-driven trajectory simplification problem can be described as follows: given a raw trajectory T and a positive integer W, the core objective of CF-TS is to generate a simplified trajectory T' such that $|T'| \leq W$, and the simplification loss $\epsilon(T')$ is minimized, where $\epsilon(T')$ can be calculated by Equations (1) and (2) according to particular application requirements.

4 Methodology

Figure 1 shows the framework of CF-TS, which is divided into two stages and the specific descriptions are as follows:

- **Coarse-grained Stage.** Given a trajectory, CF-TS first performs preliminary simplification to establish a solid foundation for subsequent refinement. Following this, a candidate point explorer is developed to further identify important points that were not selected before, with the selection criteria incorporating both distance-based metrics and directional features.
- **Fine-grained Stage.** CF-TS refines the pre-simplified trajectory by employing MCTS to iteratively evaluate and select the most promising points from the candidate set identified in the coarse-grained stage. MCTS treats trajectory simplification as a sequential decision-making process, exploring the trade-offs between effectiveness and efficiency.

4.1 Primary Salient Douglas-Peucker Initialization

As illustrated in the phase I of Figure 1, CF-TS initiates the coarse-grained simplification stage with a customized variant of DP algorithm, termed PSDP, which is based on the primary salient point concept.

Definition 4. ***(Primary Salient Point).*** *A primary salient point of a curve is the point whose perpendicular distance from the straight line connecting the start and end points exceeds a given threshold δ.*

Given a trajectory and a distance threshold δ, PSDP first connects the first point p_s and the last point p_e with a straight line, and examines the presence of a primary salient point (Definition 4) along this segment. If such a point is identified, it is retained and designated as the new start point, forming a new segment with the original end point. The process is repeated on each newly formed segment to identify additional primary salient points. If no primary salient point is detected, intermediate points are considered redundant and discarded. The procedure continues until all points have been evaluated, resulting in a simplified trajectory constructed by sequentially connecting the retained anchor points.

Unlike the traditional DP algorithm, which suffers from increasing computational cost due to multiple traversals, PSDP requires only a single trajectory traversal. Its computational complexity remains independent of the number of preserved points and is on the same scale as the original data size, making it particularly suitable for large datasets.

4.2 Candidate Point Explorer

As shown in Figure 1, following PSDP, we develop a candidate point explorer to identify relatively important points which are not included in T_c (the output of PSDP algorithm). Since PSDP primarily depends on vertical distance threshold and overlooks other essential features such as directional changes, two evaluation metrics, 1) *smoothness* and 2) *direction error*, are incorporated. Prioritizing smoothness ensures the preservation of the trajectory's structural features, while direction error ensures the retention of directional information, which are both crucial but frequently neglected for maintaining trajectory fidelity. The exploration is systematically divided into three interconnected steps as follows:

Step 1. Calculate Segment Smoothness. For each segment $\overrightarrow{p_i p_j}$ on T_c, the smoothness is defined as the ratio of the distance from p_i to p_j when traveling along the raw trajectory T and that when traveling along the anchor segment $\overrightarrow{p_i p_j}$ (Definition 2), which is formally expressed as follows:

$$Smoothness_{\overrightarrow{p_i p_j}} = \frac{\Sigma_{i \leq h < j} d\left(p_h, p_{h+1}\right)}{d\left(p_i, p_j\right)}. \tag{3}$$

Based on Equation (3), the smoothness of the corresponding segments on T measures how closely each segment of the T_c resembles a straight line after PSDP algorithm. A smoothness value close to 1 indicates that the segment is nearly straight, while a significantly larger value suggests the presence of sharp turns or substantial changes in the trajectory. Segments with higher smoothness values are prioritized for further analysis to preserve the structural integrity.

Step 2. Calculate Direction Error. We define the deviation between two directions α and β as the smallest angle of rotation from α to β or vice versa in a counterclockwise direction, expressed as follows:

$$diff\left(\alpha, \beta\right) = \min\left\{\left|\alpha - \beta\right|, 2\pi - \left|\alpha - \beta\right|\right\}. \tag{4}$$

The direction of movement from p_i to p_j in T is defined as the angle of an anticlockwise rotation from the positive X-axis to the vector $\overrightarrow{p_i p_j}$, denoted as

$dir(p_ip_j)$. Based on Equation (4), the direction error $\epsilon(\overrightarrow{p_ip_j})$ for the segment $\overrightarrow{p_ip_j}$ relative to T is calculated as the average angular differences between $dir(p_ip_j)$ and all directions within the direction set $dir[p_i : p_j]$, formulated as:

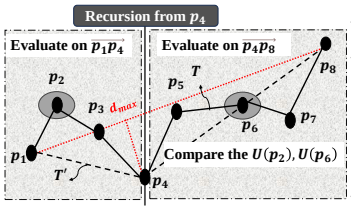
$$\epsilon(\overrightarrow{p_ip_j}) = \frac{\sum_{i \le k \le j} \text{diff}(\text{dir}(p_ip_j), \text{dir}(p_kp_{k+1}))}{|\text{dir}[p_i : p_j]|}, \tag{5}$$

where $dir[p_i : p_j]$ refers to the directional information from segments $\overrightarrow{p_ip_{i+1}}$ to $\overrightarrow{p_{j-1}p_j}$ and $|\text{dir}[p_i : p_j]|$ denotes the size of this set. The direction error, calculated by Equation (5), quantifies the deviation and identifies segments where the movement direction has been significantly altered during PSDP initial simplification. Such segments are flagged for refinement to ensure the final trajectory faithfully represents the original directional trends.

Step 3. Candidate Point Set Generation. To eliminate the impact of differing scales and ranges, the two attributes are first standardized by z-score normalization [1], denoted as S' and D' (calculated by Equations (3) to (5)), respectively. Then different weights W_s, W_d are assigned to S' and D', reflecting their relative significance, as follows:

$$SCORE = W_s \cdot S' + W_d \cdot D'. \tag{6}$$

Based on the composite score from Equation (6), the top K anchor segments with the highest scores are selected as the key segments, where K is determined by the compression ratio CR: $K = |T| \times CR$. For all points on the corresponding key segments of the original trajectory T but not selected by PSDP, the explorer adds them to the candidate point set C, preserving the significant structural and directional features of the trajectory.



Stage	$T = \langle p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8 \rangle$	Select
Coarse-grained	δ = 1, D = {0.31,0.36,1.13,0.63,0.31,0.89}	p_1, p_4, p_8
Candidate Point Set	Smoothness = {1.11, 1.25} Direction error = {0.42, 0.58}	C= $\langle p_2, p_6 \rangle$
Fine-grained	$U(p_2) = 0.69, U(p_6) = 0.73,$	p_6
Output	$T' = \langle p_1, p_4, p_6, p_8 \rangle$	

Fig. 2. Running example of CF-TS with PED.

To clearly clarify Sections 4.1 and 4.2, we present an example in Figure 2, where a trajectory $T = \{p_1, p_2, p_3, \cdots, p_8\}$ is shown at the left, with the execution process detailed on the right. The PSDP algorithm first connects p_1 and p_8 with a straight line, and calculates the perpendicular distance of each intermediate point to this line. Among these, p_4 is identified as a primary salient point since its distance $1.13 > 1$ exceeding the threshold. Consequently, p_4 is retained as a new anchor point, and PSDP applies the same process to the new segments $\overrightarrow{p_4p_8}$, resulting in the pre-simplified trajectory $T_c = \{p_1, p_4, p_8\}$. For each segment in T_c,

the candidate point explorer evaluates the smoothness and direction error of $\overrightarrow{p_1p_4}$ and $\overrightarrow{p_4p_8}$, yielding $Smoothness = \{1.11, 1.25\}, DirectionError = \{0.42, 0.58\}$. A scoring mechanism (Equation (6)) is then applied to generate the candidate point set C, identifying points p_2 and p_6 as critical for further refinement.

4.3 Fine-grained Simplification Stage

Given the sequential nature of trajectory optimization, we model the trajectory as a tree structure and utilize Monte Carlo Tree Search (MCTS) [2, 23] to efficiently explore promising points. MCTS inherently follows an exploration-exploitation strategy, prioritizing directions that lead to points with higher simplification performance. Besides, since the search space of all possible points is large, MCTS will perform on candidate point set C to identify optimal points, with each node encompassing the following key attributes:

1) State. Each node represents a sequence comprising currently selected points.

2) Node benefit. Each node ν is assigned a benefit score $\beta(\nu)$ to reflect the expected performance of a simplified trajectory passing through ν. Assume that MCTS has explored and simulated a simplified trajectory (denoted by T'_s), our objective function can be expressed as:

$$S_v = -\lambda_f f\left(T, T'_s\right) + L_{reg}\left(T'_s\right), \tag{7}$$

where $f\left(T, T'_s\right) = e^{\epsilon\left(T'_s\right)}$ (based on Equation (2)), evaluates how well T'_s resembles to T. And the regularization loss $L_{\text{reg}}\left(T'_s\right)$ is decomposed into:

$$L_{\text{reg}}\left(T'_s\right) = \lambda_{uni} L_{uni}(T'_s) + \lambda_{dir} L_{dir}(T'_s), \tag{8}$$

where λ_{uni} and λ_{dir} equal to W_s and W_d in Equation (6); L_{uni} ensures evenly spaced points and L_{dir} regularizes the angle changes in T'_s, formulated as follows:

$$\begin{aligned} L_{uni}\left(T'_s\right) &= \frac{1}{|T'_s| - 1} \sum_{i=1}^{|T'_s|-1} \left(\|p_{i+1} - p_i\| - d_{avg}\right)^2, \\ L_{dir}\left(T'_s\right) &= \frac{1}{|T'_s| - 1} \sum_{i=1}^{|T'_s|-1} \left|\text{dir}\left(p_i, p_{i+1}\right) - \text{dir}_{\text{orig}}\ \left(p_i, p_{i+1}\right)\right|, \end{aligned} \tag{9}$$

where $\|p_{i+1} - p_i\|$ is the Euclidean distance between consecutive points, d_{avg} is the average segment length, and $dir(\cdot)$ represents the directional angle of the segment. The regularization ensures spatial consistency and maintains directional integrity. Based on Equations (8) and (9), we use $-S_v$ as the maximized score $\beta(\nu)$, which would be propagated through the search tree to guide candidate point selection. For non-leaf nodes ν, the benefit score is derived from the scores of its child nodes $Ch(\nu)$ as follows:

$$\beta(v) = \frac{\sum_{v' \in Ch(v)} \beta(v')}{|Ch(v)|}. \tag{10}$$

3) Access frequency. $N(\nu)$ denotes the number of times that the node ν has been visited during the tree search through all the ways to the leaf nodes.
4) Node utility. The node utility $U(\nu)$ combines both benefit score and access frequency. Following the commonly-used upper confidence bound (UCB) [4] solution, $U(\nu)_n$ is expressed as follows:

$$U(\nu)_n = \frac{1}{n-1}\sum_{1}^{n-1}\beta(\nu) + \lambda \times \sqrt{\frac{2\ln N(\nu_f)}{N(\nu)}}, \tag{11}$$

where $N(\nu_f)$ is the total visits to the parent node of ν, and λ (typically set to 1) balances the trade-off between exploitation of past results (average benefit of $n-1$ simulations) and exploration of less-visited nodes. Equation (11) enables efficient exploration of the search space while achieving a globally optimal solution.

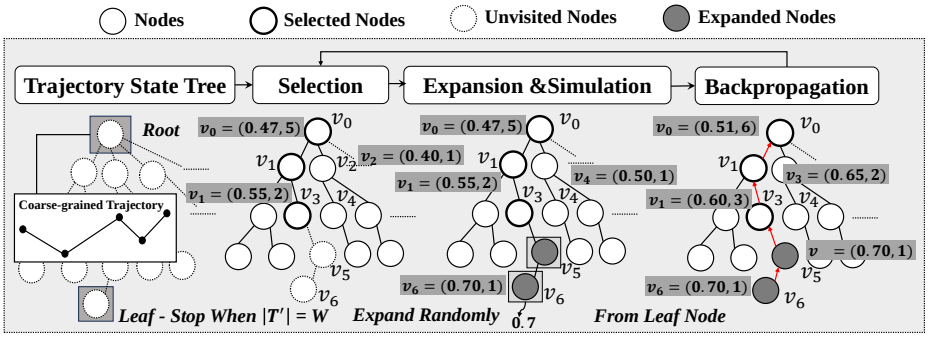


Fig. 3. The workflow of MCTS.

Figure 3 shows the workflow of MCTS, which consists of four components. For the sake of clarity, we show a couple ($U(\nu)$, $N(\nu)$) on the example for each node ν. From the root, MCTS continuously performs selection, expansion, simulation and backpropagation to update node utility for better search.

Node Selection. From the root, MCTS iteratively selects the child node with the largest utility computed by Equation (11), which continues until a node with an unexpanded child. As shown in Figure 3, the search begins at the root, where ν_1, the child with the largest utility, is selected. The corresponding point is added to T_c to form a new trajectory as a new state. Subsequently, since $U(\nu_3) > U(\nu_4)$, ν_3 is selected, and it has a node that has not been expanded, the node selection process stops and comes to the next step.

Node Expansion and Simulation. For a node ν with unexpanded children, MCTS randomly selects a child for expansion. Upon reaching the new node, a simulation step is performed to explore the rest of the tree randomly until reaching a leaf, in our case, when the trajectory compression ratio has been met. For example, if ν_5 is an unexpanded child node of ν_3, it is created, followed by ν_6, which is a leaf node corresponding to a sub-trajectory $(\nu_0, \nu_1, \nu_3, \nu_5, \nu_6)$.

Backpropagation. Following the simulation, information such as access frequency and node benefits is updated from the current node back to the root, to enhance the quality of subsequent searches. Through multiple iterations and continuous simulations, most nodes' utility values are evaluated, allowing MCTS to strategically select nodes for exploitation or exploration. This process continues until the storage budget is satisfied, progressively identifying the best points to include in the final simplified trajectory.

We continue with the example in Figure 2 to illustrate the fine-grained stage. Firstly, we model T_c as the root state, and one optional point should be added under the condition that $W = 4$. Based on the $C = \{p_2, p_6\}$, MCTS calculates the utility value for each possible state, and then point p_6 is selected. Finally, the algorithm returns the simplified trajectory $T' = \langle p_1, p_4, p_7, p_8 \rangle$.

5 Experiments

In this section, we conduct multiple experiments to verify the effectiveness of CF-TS, which are generally compliant to answer the following three questions:

1) How effective is CF-TS for the error-driven trajectory simplification compared to other baseline methods **(Q1)**?
2) What is the computational efficiency of CF-TS, and how well does it scale with increasing dataset sizes **(Q2)**?
3) How do variations in parameter settings impact the performance of CF-TS in terms of simplification effectiveness and efficiency **(Q3)**?

5.1 Experimental Setup

Datasets. We conduct the experiments on two real-world trajectory datasets, Geolife and T-Drive, both are widely used in previous TS studies [14, 28, 17]. Geolife keeps the travel records of 182 users during a period of five years. T-Drive contains trajectories from 10,357 taxis in Beijing over a week, and a detailed statistical overview of these datasets can be found in [30].

Baselines. We review the trajectory simplification evaluation work and recent literature [30, 28, 26] thoroughly so that representative error-driven oriented TS methods are included for comparison. Overall, we identify four compared baselines into two categories: **1) Non-learning-based:** ① DP-Hull [19], which achieves the same effectiveness of DP but with higher efficiency. ② Error-Search [18], which pays more attention to direction information. **2) Learning-based:** ③ RLTS [28], which utilizes RL to perform trajectory simplification. ④ S3 [7], which is a lightweight framework based on the Seq2Seq Network.

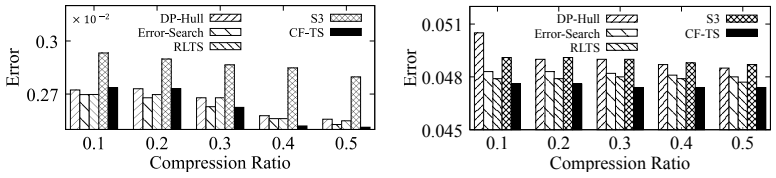
Evaluation metrics. For effectiveness, PED, SED and DAD metrics are utilized to measure the simplification error, where lower values indicate higher accuracy. Besides, the running time records the total time spent with a lower time cost reflecting better practicality and efficiency.

Parameter Settings. Following previous works [26, 30], we randomly select 1,000 trajectories from two datasets respectively to perform simplification, and

30% of which for training. For PSDP, the threshold δ is set to 10 and 300 meters for Geolife and T-Drive, respectively. For the candidate point set, the scoring function is assigned weights of 0.33 to W_s and 0.67 to W_d. The MCTS exploration is conducted with $1,000$ iterations. The error metrics employed include PED, SED and DAD with units of 10 meters, 10 meters, and 1 radian, respectively. The framework is implemented using Python.

5.2 Evaluation of Simplification Effectiveness

To answer question **Q1**, we conduct trajectory simplification on CF-TS and other four baselines under different compression rates (CR) ranging from 0.1 to 0.5. Figures 4 to 6 demonstrate the results on two datasets with three metrics respectively, and we provide observations and analyses as follows.

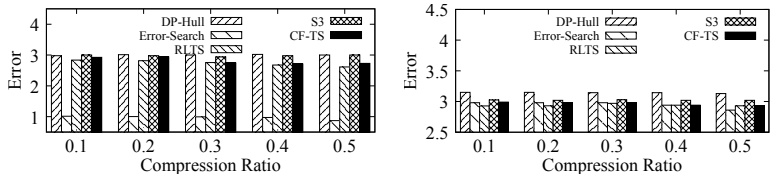


(a) Geolife Dataset (b) T-Drive Dataset

Fig. 4. Effectiveness with varying the compression ratio on PED.

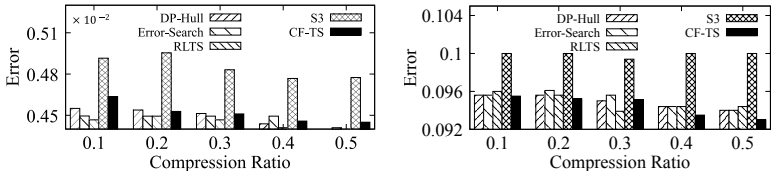
The first observation is that under all tested compression ratios, the proposed CF-TS consistently demonstrates lower average errors with PED on two datasets, significantly outperforming all the baseline methods. For example, at a CR of 0.1 on T-drive, DP algorithm yields an error of 0.0505, while S3 produces an error of 0.0491. In contrast, CF-TS reduces the error to 0.0476, representing an effectiveness improvement of 5.74% over DP and 3.05% over S3. This performance enhancement is primarily due to the optimization strategies integrated into CF-TS framework, where MCTS dynamically evaluates candidate points and iteratively refines the selection process, ensuring that the simplified trajectory retains the most critical geometric and directional features of the original data.

In terms of DAD, shown in Figure 5, Error-Search performs remarkably well on two datasets, owing to its objective function explicitly designed to minimize this metric. While CF-TS does not surpass Error-Search, it still outperforms others models by integrating distance and directional attributes when constructing the candidate point set. For example, on T-drive with a CR of 0.5, Error-Search achieves an error of 2.86, while CF-TS records a slightly higher error of 2.93. On the SED metric shown in Figure 6, CF-TS experiences a slight decrease in effectiveness but remains competitive, as the differences in SED errors of all methods are minimal. This demonstrates that CF-TS maintains robust performance even when evaluated with metrics beyond its primary optimization focus.



(a) Geolife Dataset (b) T-Drive Dataset

Fig. 5. Effectiveness with varying the compression ratio on DAD.



(a) Geolife Dataset (b) T-Drive Dataset

Fig. 6. Effectiveness with varying the compression ratio on SED.

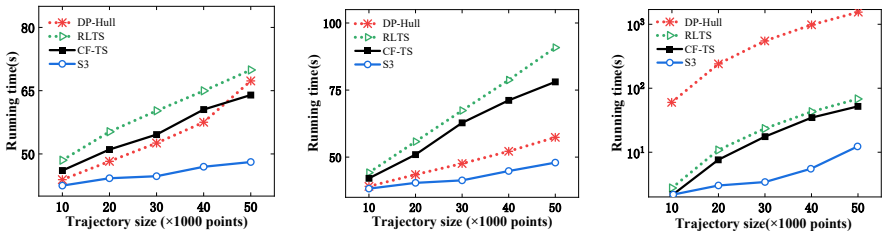
5.3 Evaluation of Simplification Efficiency

To evaluate the efficiency of CF-TS and the effect of trajectory length $|T|$ on different methods **(Q2)**, we report the results of average processing time per point as the efficiency evaluation in Table 1, Figures 7 and 8.

Table 1. Efficiency comparison with varying CR on Geolife and T-Drive datasets.

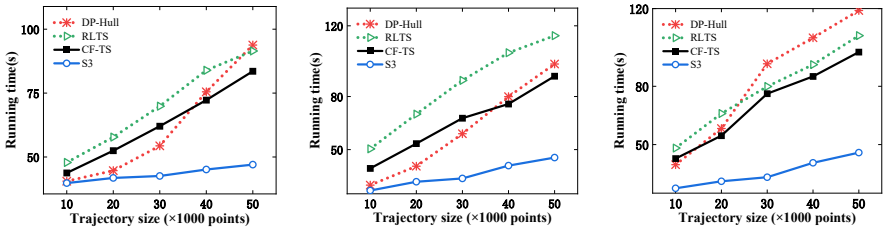
CR	Geolife				T-Drive			
	PED		DAD		PED		DAD	
	RLTS	CF-TS	RLTS	CF-TS	RLTS	CF-TS	RLTS	CF-TS
0.1	3.871	3.627	4.875	4.689	5.029	4.924	4.769	4.671
0.2	3.327	3.173	3.069	2.832	4.143	4.029	4.077	4.012
0.3	2.774	2.681	2.691	2.547	3.618	3.428	3.615	3.483
0.4	2.402	2.273	2.123	1.983	3.143	2.959	3.123	2.985
0.5	2.173	1.782	1.976	1.747	2.714	2.658	2.769	2.647

1) Overall Efficiency Comparison. As displayed in Table 1, compared to the learning-based RLTS algorithm, CF-TS shows certain advantages. ① In Geolife, CF-TS processes a single trajectory approximately 0.187 seconds faster on average than RLTS, while on the T-Drive, it achieves an average speedup of 0.191 seconds. ② The advantage becomes even more apparent under PED metric,



(a) Geolife PED Error (b) Geolife SED Error (c) Geolife DAD Error

Fig. 7. Efficiency with varying the trajectory length on Geolife.



(a) T-Drive PED Error (b) T-Drive SED Error (c) T-Drive DAD Error

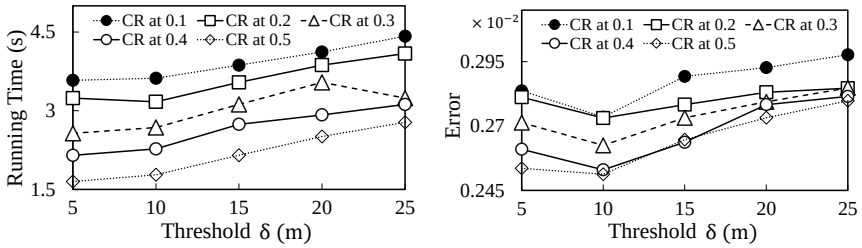
Fig. 8. Efficiency with varying the trajectory length on T-Drive.

faster about 0.21s. The efficiency improvement of CF-TS can be attributed to its initialization strategy, which serves as a warm-start for the whole algorithm, providing a more efficient basis for refinement. Additionally, CF-TS employs a two-stage simplification framework, in contrast to RLTS's single-stage approach of sequentially adding points. Despite this, CF-TS consistently outperforms RLTS in running time, highlighting its superior operational efficiency.

2) Efficiency with varying the trajectory length. we vary $|T|$ from $10,000$ to $50,000$ and randomly select 100 trajectories, with the CR of 0.3. Error-Search requires more than $10,000$ seconds in all cases, thus omitted from detailed comparison, and the experimental results are depicted in Figures 7 and 8. We can conclude that the average processing time of CF-TS slightly rises as the trajectory length increases, and this growth remains relatively moderate, demonstrating good scalability and efficiency for large-scale trajectories. Notably, CF-TS consistently outperforms RLTS and shows stable performance, especially under SED metric, achieving approximately a 12% improvement in efficiency. While marginally slower than S3 and DP-Hull due to its learning-based nature, CF-TS maintains competitive efficiency and fully meets practical requirements.

5.4 Parameter Sensitivity Analysis

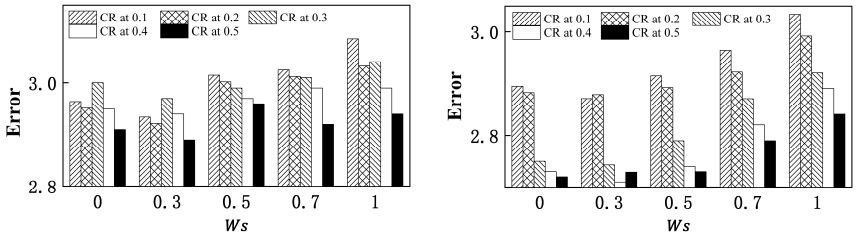
To evaluate the sensitivity of CF-TS assessing its robustness **(Q3)**, we explore key parameters and their effects on trajectory simplification performance.



(a) Running time on Geolife with PED (b) Error on Geolife with PED

Fig. 9. Effect with varying the parameter δ on Geolife.

1) The parameter δ. To study the impact of the PSDP algorithm's threshold δ on trajectory simplification, we conduct experiments with varying δ on the Geolife dataset under PED, as shown in Figure 9. For efficiency, when δ rises from 5 to 20, the running time increases from $2.57s$ to $3.24s$, reflecting a 26% increase. For effectiveness, the results in Figure 9(b) demonstrate that a threshold of 10 strikes a balance between retaining critical points in the coarse-grained stage and ensuring efficient refinement in the fine-grained stage. We can also conclude that the overall variation remains minimal despite the changes of δ, indicating that the initial threshold has a limited impact on the final simplification outcome, highlighting the robustness of the CF-TS method.



(a) Error on T-Drive with DAD (b) Error on Geolife with DAD

Fig. 10. Effect with varying the parameter W on T-Drive and Geolife.

2) The parameter W. We also investigate the impact of assigning different weights to W_s and W_d on simplification performance. As shown in Figure 10, the

results indicate that relying on a single feature leads to suboptimal simplification. Specifically, when $W_s = 0$, the DAD error on the T-Drive dataset ranges from 3.00 to 3.13, and when $W_d = 0$, the error is approximately 3.02. In contrast, when the directional error is given more weight, the DAD error decreases to $2.89 \sim 2.99$. These findings underscore the importance of considering multiple features, especially that the directional features can significantly improve performance.

6 Conclusion

This paper addresses the error-driven trajectory simplification problem, aiming to identify a sub-trajectory with a fixed number of points while minimizing error. We conclude three limitations presented in existing methods and propose a new solution, a general coarse-to-fine framework (CF-TS). In the coarse-grained stage, CF-TS employs PSDP as a warm-start to establish a robust initial set of points and constructs a candidate set by integrating multiple trajectory features. In the fine-grained stage, the trajectory is modeled as a state tree, and we employ MCTS to refine regions requiring further optimization iteratively. Extensive experiments on real-world trajectory datasets demonstrate that CF-TS achieves consistently lower compression error with competitive runtime efficiency. Future work will explore error-bounded trajectory simplification in online scenarios.

Acknowledgments. This work was supported by the Natural Science Foundation of the National Natural Science Foundation of China under grant (No. 61802273), Jiangsu Higher Education Institutions of China (No. 23KJA520011), China Science and Technology Plan Project of Suzhou (No. SYG202139).

References

1. Al Shalabi, L., Shaaban, Z., Kasasbeh, B.: Data mining: A preprocessing engine. JCST **2**(9), 735–739 (2006)
2. Bai, A., Srivastava, S., Russell, S.: Markovian state and action abstractions for mdps via hierarchical mcts. In: IJCAI. pp. 3029–3039 (2016)
3. Bellman, R.: On the approximation of curves by line segments using dynamic programming. Communications of the ACM **4**(6), 284 (1961)
4. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A survey of monte carlo tree search methods. IEEE Trans.Comput. Intell.AI Games **4**(1), 1–43 (2012)
5. Chen, C., Ding, Y., Xie, X., Zhang, S., Wang, Z., Feng, L.: Trajcompressor: An online map-matching-based trajectory compression framework leveraging vehicle heading direction and change. TITS **21**(5), 2012–2028 (2019)
6. Douglas, D.H., Peucker, T.K.: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. Cartographica **10**(2), 112–122 (1973)
7. Fang, Z., He, C., Chen, L., Hu, D., Sun, Q., Li, L., Gao, Y.: A lightweight framework for fast trajectory simplification. In: ICDE. pp. 2386–2399 (2023)
8. Han, Y., Sun, W., Zheng, B.: Compress: A comprehensive framework of trajectory compression in road networks. TODS **42**(2), 1–49 (2017)

9. Hershberger, J.E., Snoeyink, J.: Speeding up the douglas-peucker line-simplification algorithm (1992)
10. Li, T., Huang, R., Chen, L., Jensen, C.S., Pedersen, T.B.: Compression of uncertain trajectories in road networks. PVLDB **13**(7), 1050–1063 (2020)
11. Li, Z., Han, J., Ji, M., Tang, L.A., Yu, Y., Ding, B., Lee, J.G., Kays, R.: Movemine: Mining moving object data for discovery of animal movement patterns. TIST **2**(4), 1–32 (2011)
12. Lin, K., Zhao, R., Xu, Z., Zhou, J.: Efficient large-scale fleet management via multi-agent deep reinforcement learning. In: SIGKDD. pp. 1774–1783 (2018)
13. Lin, X., Ma, S., Jiang, J., Hou, Y., Wo, T.: Error bounded line simplification algorithms for trajectory compression: An experimental evaluation. TODS **46**(3), 1–44 (2021)
14. Lin, X., Ma, S., Zhang, H., Wo, T., Huai, J.: One-pass error bounded trajectory simplification. arXiv preprint arXiv:1702.05597 (2017)
15. Liu, J., Zhao, K., Sommer, P., Shang, S., Kusy, B., Jurdak, R.: Bounded quadrant system: Error-bounded trajectory compression on the go. In: ICDE. pp. 987–998 (2015)
16. Liu, J., Zhao, K., Sommer, P., Shang, S., Kusy, B., Lee, J.G., Jurdak, R.: A novel framework for online amnesic trajectory compression in resource-constrained environments. TKDE **28**(11), 2827–2841 (2016)
17. Long, C., Wong, R.C.W., Jagadish, H.: Direction-preserving trajectory simplification. PVLDB **6**(10), 949–960 (2013)
18. Long, C., Wong, R.C.W., Jagadish, H.: Trajectory simplification: On minimizing the direction-based error. PVLDB **8**(1), 49–60 (2014)
19. Marteau, P.F., Ménier, G.: Speeding up simplification of polygonal curves using nested approximations. PAA **12**(4), 367–375 (2009)
20. Meratnia, N., de By, R.A.: Spatiotemporal compression techniques for moving point objects. In: EDBT. pp. 765–782 (2004)
21. Muckell, J., Hwang, J.H., Patil, V., Lawson, C.T., Ping, F., Ravi, S.: Squish: an online approach for gps trajectory compression. In: COM.Geo. pp. 1–8 (2011)
22. Potamias, M., Patroumpas, K., Sellis, T.: Sampling trajectory streams with spatiotemporal criteria. In: SSDBM. pp. 275–284 (2006)
23. Stekovic, S., Rad, M., Fraundorfer, F., Lepetit, V.: Montefloor: Extending mcts for reconstructing accurate large-scale floor plans. In: ICCV. pp. 16034–16043 (2021)
24. Sutton, R.S.: Reinforcement learning: An introduction (2018)
25. Wang, Z., Long, C., Cong, G.: Trajectory simplification with reinforcement learning. In: ICDE. pp. 684–695 (2021)
26. Wang, Z., Long, C., Cong, G., Jensen, C.S.: Collectively simplifying trajectories in a database: A query accuracy driven approach. In: ICDE. pp. 4383–4395 (2024)
27. Wang, Z., Long, C., Cong, G., Liu, Y.: Efficient and effective similar subtrajectory search with deep reinforcement learning. arXiv preprint arXiv:2003.02542 (2020)
28. Wang, Z., Long, C., Cong, G., Zhang, Q.: Error-bounded online trajectory simplification with multi-agent reinforcement learning. In: SIGKDD. pp. 1758–1768 (2021)
29. Yang, Z., Chandramouli, B., Wang, C., Gehrke, J., Li, Y., Minhas, U.F., Larson, P.Å., Kossmann, D., Acharya, R.: Qd-tree: Learning data layouts for big data analytics. In: SIGMOD. pp. 193–208 (2020)
30. Zhang, D., Ding, M., Yang, D., Liu, Y., Fan, J., Shen, H.T.: Trajectory simplification: an experimental study and quality analysis. PVLDB **11**(9), 934–946 (2018)