Exact and Efficient Similar Subtrajectory Search: Integrating Constraints and Simplification

Liwei Deng[1], Fei Wang[1], Tianfu Wang[2], Yan Zhao[1,✉], Yuyang Xia[1], Kai Zheng[1,✉]
[1]University of Electronic Science and Technology of China, China
[2]University of Science and Technology of China, China
{deng_liwei,xiayuyang}@std.uestc.edu.cn, phinnwang@gmail.com, tianfuwang@mail.ustc.edu.cn,
yanz@cs.aau.dk, zhengkai@uestc.edu.cn

***Abstract*—Similar subtrajectory search (SimSub) aims to find a subtrajectory (i.e., a segment) from a data trajectory (the trajectory to be queried) that closely resembles the query trajectory. Compared with similar trajectory search, SimSub can capture finer-grained similarity and is vital for various trajectory analysis tasks, such as trajectory clustering and join. However, SimSub may return a subtrajectory with extremely limited length, e.g., a single point, which may not align with the expectations of real-world applications. To solve this issue, we propose a constrained SimSub (cSimSub) problem, where the length of the returned subtrajectory must be greater than or equal to a user-specified integer C. We demonstrate that this problem can be solved exactly with a time complexity equivalent to C times the complexity of the trajectory distance measurement, given that the distance function can be computed using dynamic programming (DP). We also observe that when $C = 1$, the solution of cSimSub differs from the vanilla trajectory distance computation (e.g., DTW) only in the state initialization of the DP matrix. Moreover, SimSub focuses on finding a subtrajectory with successive point indexes, which limits its applicability in certain scenarios, e.g., trajectory simplification. Thus, we extend it to sSimSub for trajectory simplification, aiming to find the most similar non-continuous subsequence of a trajectory to itself, with a length constraint of C. The subsequence, i.e., the simplified subtrajectory, obtained from sSimSub can achieve the best self-similarity. We conduct experiments on three public available datasets to demonstrate the effectiveness of the proposals. The results show that integrating sSimSub into typical query methods, e.g., KNN query, can achieve higher accuracy of these methods in simplified trajectory databases compared with other well-known trajectory simplification algorithms.**

***Index Terms*—Similar Subtrajectory Search, Dynamic Programming, Constraint, Trajectory Simplification**

I. Introduction

The prevalence of GPS-enabled devices and wireless communication technologies generates massive trajectories, which provides an opportunity for better understanding human mobility patterns and stimulating a large number of trajectory analysis tasks, such as route recommendation [1], trajectory clustering [2]–[5] and location prediction [6]–[10]. In these tasks, searching similar trajectories for a given query in a large trajectory database is an indispensable way to turn the blunt information into knowledge. However, most of the existing similar trajectory search studies treat a trajectory as a whole, which ignore the fine-grained similarity between trajectories, i.e., they fail to measure the similarity between trajectories at segment-level. To solve this problem, the similar subtrajectory search (SimSub) problem is studied recently [11], [12]. It aims to find a subtrajectory (i.e., a segment) from a given data trajectory (i.e., a trajectory to be queried), which is most similar to the query trajectory compared with the other subtrajectories in the data trajectory.

Searching similar subtrajectories is a basic unit for many applications, such as subtrajectory clustering [3]–[5], [13] and subtrajectory join [14]. A naive solution to the SimSub problem is to enumerate all possible subtrajectories from the data trajectory and then returns the most similar one to the query trajectory. Despite its simplicity, the time complexity $O(mn^3)$ is often unacceptable, where m and n denote the lengths of the query and data trajectories, respectively. To be more efficient, a previous study [11] proposes several approximation algorithms to balance the searching accuracy and efficiency. Specifically, they utilize reinforcement learning methods to examine the points in the data trajectory sequentially and determine whether to perform a split operation at each point, which achieves a time complexity of $O(mn)$. However, these methods lack a theoretical guarantee regarding the accuracy of the searched subtrajectory. A recent study [12] proposes an exact algorithm, called conversion-matching algorithm (CMA), to solve the SimSub problem. Its main idea is to find the optimal subtrajectory by computing the minimum cost of converting the query trajectory into the data trajectory, which achieves the time complexity of $O(mn)$ for many trajectory distance functions, such as weighted edit distance (WED) and dynamic time warping (DTW). Despite its efficiency in solving the SimSub problem, we notice that the length of the returned subtrajectory can be extremely limited, e.g., one point, which may not be expected for many real applications, e.g., subtrajectory clustering. To solve this problem, we extend the SimSub problem to a more general case, i.e., constrained SimSub (cSimSub), where a user can specify a minimal length C of the returned subtrajectory. It should be noted that when C is set to 1, cSimSub will

✉ Corresponding authors: Yan Zhao and Kai Zheng. Yan Zhao is with Shenzhen Institute for Advanced Study, University of Electronic Science and Technology of China. Kai Zheng is with Yangtze Delta Region Institute (Quzhou), and School of Computer Science and Engineering, University of Electronic Science and Technology of China.

degenerate to the original SimSub problem. To exactly solve the cSimSub problem, we propose an algorithm based on dynamic programming (DP) and theoretically show that our method works in a time complexity of $O(Cmn)$ as long as the trajectory distance can be implemented in a dynamic programming way.

Furthermore, in practice, two challenging concerns in trajectory analysis are storing [15]–[20] and query [21]. These problems can be effectively solved by trajectory simplification, where the non-informative points in the original trajectory are deleted. However, most of the existing studies ignore the effect of trajectory simplification in terms of query accuracy, i.e., queries run on a simplified database may yield different results compared to queries on the original database. In this study, we extend the SimSub problem to sSimSub for trajectory simplification, which aims to find the most similar subsequence of a trajectory to itself, with a specified length C. Through the preservation of self-similarity, we empirically show that solving sSimSub yields superior query accuracy for KNN queries compared to previous trajectory simplification algorithms like Top-Down [22], Bottom-Up [23], and RLTS [24].

To summarize, we make the following contributions:

- We extend the SimSub problem to encompass a more general case, termed cSimSub, and propose an exact algorithm to solve it. We theoretically show that our method operates with a time complexity of $O(Cmn)$ for the majority of trajectory distance metrics, provided these metrics can be implemented in a dynamic programming way.
- We formulate a new problem definition, called sSimSub, aiming at performing trajectory simplification. By solving the sSimSub problem, we aim to derive a simplified trajectory that closely resembles to the original trajectory.
- We conduct extensive experiments on three publicly available datasets to verify the superiority of our proposed methods. The results show that solving sSimSub for trajectory simplification leads to better query accuracy in KNN query experiments.

Organization. We review the related work in Section II and provide preliminaries and the problem definition in Section III. Section IV presents our algorithms to solve the cSimSub and sSimSub problem. We report the experimental results in Section V and conclude this paper in Section VI.

II. Related Work

Trajectory Similarity Measurement. Measuring the similarity between trajectories is a crucial aspect of various trajectory analysis tasks and applications [25]–[33], which is to effectively determine the resemblance between two trajectories [34]. Traditionally, classical solutions follow a matching-and-measurement paradigm, such as DTW [35], LCSS [36], EDR [37], and Frechet [38]. These methods aim to find the optimal alignment through dynamic programming and quantify the dissimilarity between corresponding points. Recently, learning-based algorithms are proposed to accelerate similarity computation [34], [39]–[47]. For example, Yao et al. [48] utilize deep-metric-learning to approximate existing similarity measures. Fang et al. [49] employ deep attentive networks to learn a spatio-temporal trajectory similarity metric in road networks. Han et al. [44] propose GTS that enhances the learned embeddings on the road network with POI information and GNN. Zhang et al. [45] propose Traj2SimVec, which incorporates the matching information of the ground-truth measurement as an auxiliary supervision to train a better neural networks. In this study, we only focus the *exact* solution of the SimSub problem and its variants. Thus, we only utilize the classical measurements computed using dynamic programming as our distance functions.

Subtrajectory Search. A previous study [50] focuses on finding the most similar subtrajectory from a whole database. They propose a two-phases algorithm comprising filtering and verification. The filtering phase prunes trajectories whose distance from the query trajectory exceeds a predefined threshold, while the verification phase identifies the most similar subtrajectory from each searched trajectory. It should be noted that all candidate subtrajectories within the searched trajectory need to be checked. Another line of research [11], [12], [51] aims to accelerate the verification phase, where given a query and data trajectory, the goal is to find the most similar subtrajectory from the data trajectory. For example, Wang et al. [11] propose algorithms, e.g., ExactS, RLS, and RLS-Skip, to speed up the search process within a data trajectory. Specifically, ExactS enumerates all possible candidate subtrajectories and incrementally computes the similarity of subtrajectories with the same prefix, which can solve the SimSub problem exactly with a time complexity of $O(mn^2)$. RLS and RLS-Skip are approximation algorithms that utilize reinforcement learning to select appropriate splitting points, enhancing search efficiency. Recently, Jin et al. [12] propose a conversion-matching algorithm to exactly solve the SimSub problem with time complexity of $O(mn)$ for many trajectory distance measurements, such as WED and DTW, which aims to find the conversion-matching path with minimum cost for converting the query trajectory into the data trajectory. However, CMA is not applicable to certain trajectory distance measurements like LCSS and LORS and needs to define numerous transition cost functions for each measurement. Furthermore, despite significant efforts in subtrajectory search, we should note that existing solutions of the SimSub problem do not guarantee the quality of the returned subtrajectory in terms of its length.

Trajectory Simplification. The increasing volume of trajectory data introduces the challenges in terms of storing [15]–[17] and query [21]. Trajectory simplification [21]–[24], [52] offers a solution by dropping points from trajectories, thus lowering the storage cost and expediting query processing. Existing methods typically assume a storage budget and aim to produce simplified trajectories to minimize the difference from the original trajectory [21]. For example, Top-Down [22] starts from the first and last points and then repeatedly inserts points with the greatest error, measured by distance functions, such as SED [53], [54], PED [55], [56], and DAD [57], [58], until the predefined budget is exhausted. Conversely, Bottom-Up [23] scans all points of the input trajectory, iteratively discarding

TABLE I
NOTATION TABLE

Notation	Explanation
p_i, q_i	A trajectory sample point
$E(q_i, p_j)$	The Euclidean distance between two sample points q_i and p_j
T	A trajectory
T_q	A query trajectory with length m
T_d	A data trajectory with length n
$T[i, j]$	A subtrajectory starts from i and ends at j
$T[i_1, i_2, \cdots, i_k]$	A subsequence of trajectory T with length k, in which points p_{i_1}, $\cdots$, and p_{i_k} are included
$\Theta(\cdot, \cdot)$	An abstract trajectory distance or similarity measurement, which can be instantiated with a concrete measurement, such as DTW, Frechet, and LCSS
C	A user-specified positive integer
P	A matrix in $\mathcal{R}^{m \times n}$ to record the starting point for each cell of the DP matrix

points with the smallest error. Recently, with the prevalence of reinforcement learning (RL), advanced approaches based on RL have emerged [21], [24], [59]. For example, Wang et al. [24] adopt the Bottom-Up strategy and discard points based on a learned policy rather than relying on heuristic rules as seen in previous studies. Despite its effectiveness in reducing storage cost, trajectory simplification can compromise query accuracy; for instance, the results of KNN query from original and simplified databases often differ. Few studies consider query accuracy when performing trajectory simplification. In this study, we extend the SimSub problem to the sSimSub problem for trajectory simplification. Solving the sSimSub problem aims to optimize the similarity between the simplified trajectory and its original trajectory. Notably, our empirical findings demonstrate that maintaining self-similarity can achieve better query accuracy in terms of KNN queries compared with classical trajectory simplification methods.

III. PRELIMINARIES

We first formalize the problem of cSimSub and sSimSub and then introduce trajectory similarity measurement that we exploit in the proposed framework. To make the content more clear, we conclude the main notations used in this paper in Table I.

A. Problem Statement

Definition 1 (Trajectory)**.** *A trajectory T with length n is a sequence of points, denoted as $<p_1, p_2, \cdots, p_n>$, in which $p_i = (x_i, y_i, t_i)$ means that the location is (x_i, y_i) at time t_i.*

Definition 2 (Subtrajectory)**.** *Given a trajectory T and $1 \leq i \leq j \leq n$, a subtrajectory is denoted as $T[i, j]$, which is a portion of T that starts from the i-th point and ends at the j-th point, i.e., $T[i, j] =< p_i, p_{i+1}, \cdots, p_j >$.*

Problem 1 (Similar Subtrajectory Search (SimSub) [11])**.** *Given a data trajectory (the trajectory to be queried) T_d with length n and a query trajectory T_q with length m, the SimSub problem is to find a subtrajectory of T_d, denoted as $T[i^*, j^*]$ $(1 \leq i^* \leq j^* \leq n)$, which is most similar to T_q according to a trajectory distance measure $\Theta(\cdot, \cdot)$, i.e., $T[i^*, j^*] = \arg\min_{1 \leq i \leq j \leq n} \Theta(T_q, T_d[i, j])$.*

From the definition of SimSub, we can observe that there is no quality guarantee in terms of the length of the resulting subtrajectory. Thus, the subtrajectory returned could consist of just one point. This may be unexpected for some applications. To deal with this situation, we extend the SimSub problem to a more general case as follows.

Problem 2 (Constrained SimSub (cSimSub))**.** *Given a data trajectory T_d, a query trajectory T_q, and a user-specified integer C, the cSimSub problem is to find a subtrajectory, $T[i^*, j^*]$, which is most similar to T_q according to a trajectory distance measure $\Theta(\cdot, \cdot)$, in which $j^* - i^* + 1 \geq C$. It should be noted that when C equals 1, the cSimSub problem will reduce to the SimSub problem.*

An intuitive solution to answer a user's query to the whole database is to scan the data trajectories, solve the SimSub problem for each data trajectory, and update the most similar subtrajectory found so far. This solution can be improved through incorporating advanced indexing techniques [60], [61], e.g., Grid-Based Pruning (GBP) [12] and Key Points Filter (KPF) [12], to prune unpromising trajectories. It should be noted that these pruning techniques are independent of the solution to the SimSub problem. In this study, our focus lies on solving the SimSub and its variants without developing pruning algorithms.

Indeed, the SimSub problem can be regarded as a special case of trajectory simplification, where both T_q and T_d represent the same trajectory T to be compressed. The returned subtrajectory can be treated as the simplified version of the original trajectory. However, there are two issues from the original SimSub problem if it is applied to trajectory simplification. First, the subtrajectory derived from SimSub is a successive portion of the entire trajectory, leading to a significant loss of information. Second, trajectory simplification cannot be achieved through solving the origin SimSub problem since it would simply return the entire trajectory T itself, meaning that the most similar subtrajectory of T is T. To achieve the goal of simplification, we modify the SimSub problem as follows.

Definition 3 (Subsequence)**.** *Given a trajectory T with length n and $1 \leq i_1 < i_2 < \cdots < i_k \leq n$, a subsequence with length k is denoted as $T^s[i_1, \cdots, i_k] =< p_{i_1}, p_{i_2}, \cdots, p_{i_k} >$. It should be noted that the indexes do not necessarily need to be consecutive. For example, p_{i_2} and p_{i_1+1} may not be the same point, meaning that i_2 may not be equal to $i_1 + 1$.*

When $< p_{i_1}, p_{i_2}, \cdots, p_{i_k} >$ is clear from the context, we use T^s to denote $T^s[i_1, \cdots, i_k]$.

Problem 3 (SimSub for Trajectory Simplification (sSimSub))**.** *Given a compression ratio r and a trajectory T with length N to be compressed, the sSimSub problem is to find a subsequence T^s with length C, where $C = \lfloor N * r \rfloor$ and T^s is the most similar subsequence to T according to tra-*

jectory distance measurement $\Theta(\cdot,\cdot)$*, i.e.,* $T^s[i_1,\cdots,i_C] = \arg\min_{1\le i_1<\cdots<i_C\le n}\Theta(T,T^s)$*. Moreover, inspired by previous trajectory compression studies [22]–[24], the first and the last points are preserved in the compressed trajectory, i.e.,* $i_1 = 1$ *and* $i_C = n$*.*

According to the definition of sSimSub, it is obvious that the compressed trajectory T_s obtained by solving the sSimSub problem can achieve optimal self-similarity under the user-specified similarity measurement. In this study, we empirically investigate whether preserving self-similarity in trajectory compression can enhance query accuracy, specifically in terms of KNN queries. A KNN query takes a query trajectory T_q as a parameter and returns a set of k trajectories (denoted by R) such that $\forall T_i \in R$, $\forall T_j \in D - R$, $\Theta(T_q,T_i) \le \Theta(T_q,T_j)$, where $\Theta(\cdot,\cdot)$ represents a trajectory distance measure.

B. Trajectory Similarity Measurement

The SimSub problem and its variants assume an abstract trajectory distance measurement, in which any existing measurements can be applied. In this study, we only consider traditional measurements that can be computed through dynamic programming, such as DTW [35], LCSS [36], Frechet [38], and EDR [37]. To make the whole manuscript self-contained, we briefly introduce three well-known trajectory distance measurements as follows.

DTW [35]. Given a query trajectory $T_q =< q_1,q_2,\cdots,q_m >$ and a data trajectory $T_d =< p_1,p_2,\cdots,p_n >$, the DTW distance is defined as below:

$$D_{i,j} = \begin{cases} \sum_{h=1}^{j} E(q_1,p_h) & if\ i=1 \\ \sum_{h=1}^{i} E(q_h,p_1) & if\ j=1 \\ E(q_i,p_j) + \min\{D_{i-1,j-1}, D_{i-1,j}, D_{i,j-1}\} & otherwise \end{cases} \tag{1}$$

where $D_{i,j}$ represents the DTW distance between $T_q[1,i]$ and $T_d[1,j]$, and $E(q_i,p_j)$ is the Euclidean distance between points q_i and p_j. Usually, $E(\cdot,\cdot)$ can be computed in time complexity of $O(1)$. Thus, the time complexity of DTW is $O(mn)$.

Frechet [38]. Given a query trajectory T_q and a data trajectory T_d, the Frechet distance is defined as below:

$$D_{i,j} = \begin{cases} \max_{h=1}^{j} E(q_1,p_h) & if\ i=1 \\ \max_{h=1}^{i} E(q_h,p_1) & if\ j=1 \\ \max\{E(q_i,p_j), \min\{D_{i-1,j-1}, D_{i-1,j}, D_{i,j-1}\}\} & otherwise \end{cases} \tag{2}$$

where $D_{i,j}$ represents the Frechet distance between $T_q[1,i]$ and $T_d[1,j]$. The time complexity of Frechet is equivalent to that of DTW.

LCSS [62]–[64]. Given a query trajectory $T_q =< q_1,q_2,\cdots,q_m >$, a data trajectory $T_d =< p_1,p_2,\cdots,p_n >$ and a threshold ϵ, the LCSS similarity is defined as below:

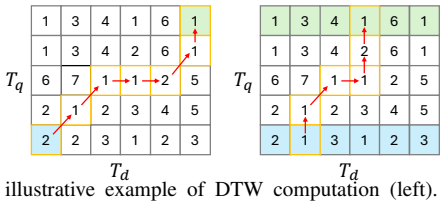

Fig. 1. An illustrative example of DTW computation (left). The SimSub problem under the DTW measurement (right).

$$D_{i,j} = \begin{cases} \max_{h=1}^{j} \mathbb{I}_{1,h} & if\ i=1 \\ \max_{h=1}^{i} \mathbb{I}_{h,1} & if\ j=1 \\ D_{i-1,j-1}+1 & if\ j>1, i>1, \mathbb{I}_{i,j}=1 \\ \max\{D_{i-1,j}, D_{i,j-1}\} & if\ j>1, i>1, \mathbb{I}_{i,j}=0 \end{cases} \tag{3}$$
$$where\ \mathbb{I}_{i,h}=1, if\ E(q_i,p_h)\le\epsilon,\ otherwise\ \mathbb{I}_{i,h}=0$$

where $D_{i,j}$ represents the LCSS similarity between $T_q[1,i]$ and $T_d[1,j]$, and $E(q_i,p_j)$ is the Euclidean distance between point q_i and p_j. It should be noted that this definition is similarity rather than distance, i.e., larger value of LCSS, more similar between two trajectories. We can follow the previous study [65] to convert the similarity into the distance as follows:

$$D(T_1,T_2) = 1 - \frac{D_{m,n}}{m+n-D_{m,n}} \tag{4}$$

IV. Methodology

In this section, we first present an exact solution for SimSub, followed by extensions of this solution to address cSimSub and sSimSub. For clarity, we adopt DTW to measure the distance between query trajectory T_q and data trajectory T_d in the SimSub problem as an example. Then, we show that our algorithm works for almost all the trajectory distance measurements as long as they can be computed through dynamic programming.

A. An Exact Solution of SimSub

To better understand the to-be-solved problem and our proposed solution, we first provide a perspective of DTW computation from successive point-matching.

Definition 4 (Point-Matching Sequence (PMS))**.** *Given a query trajectory* T_q *with length* m*, a data trajectory* T_d *with length* n*,* $1 \le i_1 \le i_2 \cdots \le i_z \cdots \le m$*, and* $1 \le j_1 \le j_2 \cdots \le j_z \cdots \le n$*, a point-matching sequence, i.e.,* $PMS = [(i_1,j_1),(i_2,j_2),\cdots,(i_z,j_z),\cdots]$*, is to describe the point-matching situation between trajectory* T_q *and* T_d*, in which* (i_z,j_z) *indicates* $T_q[i_z]$ *and* $T_d[j_z]$ *is matched.*

It should be noted that Definition 4 is a general definition of point-matching, which has no extra constraint on the matched points.

Definition 5 (Successive Point-Matching Sequence (SPMS))**.** *Successive point-matching sequence is a special PMS, where* $i_{k+1} - i_k \le 1$ *and* $j_{k+1} - j_k \le 1$*.*

With the above definitions, DTW can be understood from the perspective of successive point-matching. DTW is to find the optimal successive point-matching sequence,

i.e., $SPMS = [(i_1, j_1), (i_2, j_2), \cdots (i_L, j_L)]$, such that $\sum_{z\in[1,L]} E(q_z, p_z)$ is minimized, where $i_1 = j_1 = 1$, $i_L = m$ and $j_L = n$. An example of DTW computation can be seen in Figure 1, where the value in each cell represents the Euclidean distance between two matched points. DTW is to find a successive path from the bottom-left (i.e., the blue cell) to the upper-right (i.e., the green cell) to minimize the sum of the matching distances.

Considering a subtrajectory such as $T_d[2, 4] = (p_2, p_3, p_4)$ in this example, the DTW distance between T_q and $T_d[2, 4]$ is to find a successive path from cell $[1, 2]$ to cell $[5, 4]$ with the minimal sum of the point distances as shown in the left of Figure 1. When all possible subtrajectories in T_d are enumerated, the SimSub problem under the measurement of DTW can be reformulated as follows:

Problem 4 (SimSub with DTW Distance)**.** *The SimSub problem with the DTW distance is to find the optimal successive point-matching sequence $SPMS$, such that $\sum_{z\in[1,L]} E(q_z, p_z)$ is minimized, where $i_1 = 1$ and $i_L = m$.*

In contrast to the conventional DTW, the key distinction in Problem 4 lies in the flexibility of choosing the starting and ending cells from any position in the first and last rows, respectively. An example is depicted on the right side of Figure 1. The most similar subtrajectory can be identified by starting from the second point and ending at the fourth point in the data trajectory T_d.

From Problem 4, it is obvious that this problem can be solved through dynamic programming since the searched subpath can be reused. Inspired by the state transformation equations in DTW as shown in Equation 1, we define the state of dynamic programming to solve SimSub as follows:

Definition 6 (State of DP for Solving SimSub)**.** *Given a query trajectory T_q, a data trajectory T_d, and a trajectory distance measurement $\Theta(\cdot, \cdot)$, $S_{i,j}$ is the state of DP to solve the SimSub problem, which represents the minimal distance between $T_q[1, i]$ and any subtrajectory ending with $T_d[j]$, i.e., $S_{i,j} = \min_{z\in[1,j]} \Theta(T_q[1, i], T_d[z, j])$.*

Solution 1 (DP-based Solution for the SimSub problem with DTW distance)**.** *Based on Definition 6, SimSub's state transition equation with DTW distance can be expressed as follows:*

$$S_{i,j} = \begin{cases} E(q_1, p_j) & if\ i = 1 \\ \sum_{h=1}^{i} E(q_h, p_1) & if\ j = 1 \\ E(q_i, p_j) + \min\{S_{i-1,j-1}, S_{i-1,j}, S_{i,j-1}\} & otherwise \end{cases} \quad (5)$$

When $S_{i,j}$ is completely computed, the end index $j*$ of the most similar subtrajectory can be obtained through $j* = \arg\min_j S[m, j]$. The starting index $i*$ can be traced through the reverse process starting from $S[m, j*]$ as shown in Algorithm 1.

Algorithm 1: Tracing the most similar subtrajectory from the DP matrix under DTW distance

Input: The DP matrix $S \in \mathcal{R}^{m\times n}$ and the data trajectory T_d
Output: The most similar subtrajectory $T_d[i^*, j^*]$

```
j* ← arg min_j S[m, j];
i ← m, j ← j*;
while i > 1 and j > 1 do
    z ← arg min{S_{i,j-1}, S_{i-1,j-1}, S_{i-1,j}};
    if z == 0 then
        j ← j − 1;
    end
    else if z == 1 then
        i ← i − 1;
        j ← j − 1;
    end
    else
        i ← i − 1;
    end
end
i* ← j;
return T_d[i*, j*];
```

Proof. When i equals 1,

$$\begin{aligned} S_{1,j} &= \min_{z\in[1,j]} DTW(T_1[1, 1], T_d[z, j]) \\ &= \min_{z\in[1,j]} \sum_{h=z}^{j} E(q_1, p_h) = E(q_1, p_j) \end{aligned} \quad (6)$$

When j equals 1,

$$\begin{aligned} S_{i,1} &= \min_{z\in[1,j]} DTW(T_q[1, i], T_d[1, 1]) \\ &= DTW(T_q[1, i], T_d[1, 1]) = \sum_{h=1}^{i} E(q_h, p_1) \end{aligned} \quad (7)$$

For the other situations,

$$\begin{aligned} S_{i,j} &= \min_{z\in[1,j]} DTW(T_q[1, i], T_d[z, j]) \\ &= \min\{DTW(T_q[1, i], T_d[j, j]), \min_{z\in[1,j-1]} DTW(T_q[1, i], T_d[z, j])\} \\ &= E(q_i, p_j) + \min\{\min_{z\in[1,j]} DTW(T_q[1, i-1], T_d[z, j]), \\ &\quad \min_{z\in[1,j-1]} DTW(T_q[1, i], T_d[z, j-1]), \\ &\quad \min_{z\in[1,j-1]} DTW(T_q[1, i-1], T_d[z, j-1])\} \\ &= E(q_i, p_j) + min\{S_{i-1,j}, S_{i,j-1}, S_{i-1,j-1}\} \end{aligned} \quad (8)$$

From the formulation of the Equations 1 and 5, an interesting fact we can observe is that the only difference between the DTW computation and the SimSub computation under the DTW distance is the computation of the elements in the first row, i.e., when i equals 1. This phenomenon can also be observed when employing other trajectory distance measurements, e.g., Frechet, in the SimSub problem, as below.

Solution 2 (DP-based Solution for SimSub problem with Frechet distance)**.** *Based on the state definition above, the state transition equation of SimSub with Frechet distance can be formulated as follows:*

$$S_{i,j} = \begin{cases} E(q_1, p_h) & if\ i = 1 \\ \max_{h=1}^{i} E(q_h, p_1) & if\ j = 1 \\ \max\{E(q_i, p_j), \min\{S_{i-1,j-1}, S_{i-1,j}, S_{i,j-1})\}\} & \\ otherwise & \end{cases} \quad (9)$$

The intrinsic reason of this phenomenon is that, in contrast to the DP-based trajectory distance measurement $\Theta(\cdot,\cdot)$, the SimSub problem under $\Theta(\cdot,\cdot)$ only changes the starting cell (cf. Figure 1).

When LCSS measurement is adopted in the SimSub problem, intuitively, the computation of solving the SimSub problem should be the same as the calculation of LCSS without any modification since this measurement is defined as the longest common subsequence. However, this intuition is not theoretically proved before. To bridge this gap, in this section, we will provide strict proof to demonstrate this intuition. In the following, we obtain the solution based on the similarity version of LCSS (cf. Equation 18) since the distance conversion can be easily obtained after the DP procedure is completed. To make our algorithm work in the similarity condition, we modify our original state definition (cf. Definition 6).

Definition 7 (State of DP to Solve SimSub under LCSS measurement)**.** *Given a query trajectory T_q, a data trajectory T_d and a trajectory similarity measurement $\Theta(\cdot,\cdot)$, $S_{i,j}$ is the state of DP to solve SimSub problem, which represents the **maximal similarity** between $T_q[1,i]$ and any subtrajectory ending with $T_d[j]$, i.e., $S_{i,j} = \max_{z\in[1,j]} \Theta(T_q[1,i], T_d[z,j])$.*

Solution 3 (DP-based Solution for SimSub problem with LCSS measurement)**.** *Based on the state definition above, the state transition equation of SimSub with LCSS similarity can be expressed as follows:*

$$S_{i,j} = \begin{cases} \max\limits_{h=1}^{j} \ \mathbb{I}_{1,h} & if\ i = 1 \\ \max\limits_{h=1}^{i} \ \mathbb{I}_{h,1} & if\ j = 1 \\ S_{i-1,j-1} + 1 & if\ j>1, i>1, \mathbb{I}_{i,j} = 1 \\ \max\{S_{i-1,j}, S_{i,j-1}\} & if\ j>1, i>1, \mathbb{I}_{i,j} = 0 \end{cases} \tag{10}$$
$$where\ \mathbb{I}_{i,h} = 1, if\ E(q_i, p_h) \le \epsilon,\ otherwise\ \mathbb{I}_{i,h} = 0$$

When $\forall_{i,j} S_{i,j}$ is completely computed, the end index $j*$ of the most similar subtrajectory can be obtained through $j* = \arg\max_j S[n,j]$. The start index $i*$ can be traced through the reverse process starting from $S[n, j*]$.

Proof. When $i = 1$,

$$\begin{aligned} S_{i,j} &= \max_{z\in[1,j]} LCSS(T_q[1,1], T_d[z,j]) \\ &= \max_{z\in[1,j]} \max_{h=z}^{j} \mathbb{I}_{1,h} \\ &= \max_{h=1}^{j} \mathbb{I}_{1,h} \end{aligned} \tag{11}$$

When $j = 1$,

$$\begin{aligned} S_{i,j} &= LCSS(T_q[1,i], T_d[1,1]) \\ &= \max_{h=1}^{i} \mathbb{I}_{h,1} \end{aligned} \tag{12}$$

When $j>1, i>1, \mathbb{I}_{i,j} = 1$,

$$\begin{aligned} S_{i,j} &= \max_{z\in[1,j]} LCSS(T_q[1,i], T_d[z,j]) \\ &= \max\{\max_{z\in[1,j-1]} LCSS(T_q[1,i-1], T_d[z,j-1]) + 1, \\ &\quad LCSS(T_q[1,i], T_d[j,j])\} \\ &= \max\{S_{i-1,j-1} + 1, LCSS(T_q[1,i], T_d[j,j])\} \\ &= S_{i-1,j-1} + 1 \end{aligned} \tag{13}$$

The equation above from the penultimate line to the last line is based on a fact that: $LCSS(T_q[1,i], T_d[j,j]) \le 1 \le S_{i-1,j-1} + 1$. When $j>1, i>1, \mathbb{I}_{i,j} = 0$,

$$\begin{aligned} S_{i,j} &= \max_{z\in[1,j]} LCSS(T_q[1,i], T_d[z,j]) \\ &= \max\{\max_{z\in[1,j-1]} \max(LCSS(T_q[1,i-1], T_d[z,j]), \\ &\quad LCSS(T_q[1,i], T_d[1,j-1])), LCSS(T_q[1,i], T_d[j,j])\} \\ &= \max\{\max(S_{i-1,j}, S_{i,j-1}), LCSS(T_q[1,i], T_d[j,j])\} \\ &= \max\{S_{i-1,j}, S_{i,j-1}, LCSS(T_q[1,i-1], T_d[j,j])\} \\ &= \max\{S_{i-1,j}, S_{i,j-1}\} \end{aligned} \tag{14}$$

The penultimate line above is obtained from the condition $\mathbb{I}_{i,j} = 0$. The last line is induced from $LCSS(T_q[1,i-1], T_d[j,j]) \le S_{i-1,j}$. □

B. An Exact Solution to cSimSub

Since the quality of the returned subtrajectory in terms of length cannot be guaranteed through solving the SimSub problem, a new problem called cSimSub is proposed, in which the length of the returned subtrajectory is no less than a user-specified integer C. To solve this problem, in this section, we extend the solution for SimSub and redefine the DP state.

Definition 8 (State of DP for Solving cSimSub)**.** *Given a query trajectory T_q, a data trajectory T_d, and a trajectory distance measurement $\Theta(\cdot,\cdot)$, $S_{i,j,k}$ is the state of DP for solving the cSimSub problem, which represents the minimal distance between $T_q[1,i]$ and any subtrajectory whose length is no less than a positive integer k ending with $T_d[j]$, i.e., $S_{i,j,k} = \min_{z\in[1,j-k+1]} \Theta(T_q[1,i], T_d[z,j])$.*

Solution 4 (DP-based Solution for the cSimSub problem with DTW distance)**.** *It is obvious that when k equals 1, the cSimSub problem degenerates to the SimSub problem, which can be solved as stated above. In this solution, we only need to consider the situation when $k \ge 2$. Based on the state definition above, the state transition equation for cSimSub with DTW distance can be represented as follows:*

$$S_{i,j,k} = \begin{cases} E(q_1, p_j) + S_{i,j-1,k-1} & if\ i = 1, j \ge k \\ +\infty & if\ j < k \\ E(q_i, p_j) + \min\{S_{i-1,j,k}, S_{i-1,j-1,k-1}, S_{i,j-1,k-1}\} & otherwise \end{cases} \tag{15}$$

Once all elements of $S_{i,j,k}$ are completely computed (i.e., $k \in [1, C]$), the end index $j*$ of the most similar subtrajectory can be obtained through $j* = \arg\min_j S_{m,j,C}$. The start index $i*$ can be traced in a similar manner to the solution of the SimSub problem.

Proof. When $i = 1$ and $j \ge k$,

$$\begin{aligned} S_{i,j,k} &= \min_{z\in[1,j-k+1]} DTW(T_q[1,1], T_d[z,j]) \\ &= DTW(T_q[1,1], T_d[j-k+1,j]) \\ &= E(q_1, p_j) + DTW(T_q[1,1], T_d[j-k+1, j-1]) \\ &= E(q_1, p_j) + \min_{z\in[1,j-k]} DTW(T_q[1,1], T_d[z, j-1]) \\ &= E(q_1, p_j) + S_{i,j-1,k-1} \end{aligned} \tag{16}$$

When $j < k$, the subtrajectroy ending with j does not satisfy the length constraint. Thus, we directly initialize these elements as an infinite value. Otherwise, we have:

$$\begin{aligned} S_{i,j,k} &= \min_{z\in[1,j-k+1]} DTW(T_q[1,i], T_d[z,j]) \\ &= E(q_i,p_j) + \min\{\min_{z\in[1,j-k+1]} DTW(T_q[1,i-1], T_d[z,j]), \\ &\quad \min_{z\in[1,j-k+1]} DTW(T_q[1,i-1], T_d[z,j-1]), \\ &\quad \min_{z\in[1,j-k+1]} DTW(T_q[1,i], T_d[z,j-1])\} \\ &= E(q_i,p_j) + \min\{S_{i-1,j,k}, S_{i-1,j-1,k-1}, S_{i,j-1,k}\} \end{aligned} \tag{17}$$

□

Example 1. *Figure 2 depicts a running example of the solution for the cSimSub problem under DTW measurement when k varies from 1 to 2, where N indicates the positive infinite value. For the state $S_{3,2,2}$, when performing the state transition from left and left-down, it obtains the value from previous state matrix, i.e., $S_{3,1,1}$ and $S_{2,2,1}$,* ***since the length of the matched subtrajectory will increase by 1 with these two directions****. When performing the state transition from down, it obtains the value from current state matrix, i.e., $S_{3,1,2}$. Then, according to the definition of DTW, i.e., it minimizes the* ***sum*** *of the point-matching distances (see Problem 4), $S_{3,2,2}$ is obtained according to the equation: $S_{3,2,2} = E(T_q[3], T_d[2]) + \min\{S_{3,1,1}, S_{2,2,1}, S_{3,1,2}\} = 7 + \min\{10, 4, 3\} = 10$.*

The solution for other trajectory distance measurements can be derived in a similar way, which is omitted for its simplicity. When similarity measurement is considered, e.g., LCSS, the state of cSimSub should be slightly modified. We provide the example of LCSS as follows.

Definition 9 (State of DP to Solve cSimSub under LCSS measurement)**.** *Given a query trajectory T_q, a data trajectory T_d and a trajectory similarity measurement $\Theta(\cdot,\cdot)$, $S_{i,j,k}$ is the state of DP to solve cSimSub problem, which represents the maximal similarity between $T_q[1,i]$ and any subtrajectory whose length is no less than a positive integer k and ending with $T_d[j]$, i.e., $S_{i,j,k} = \max_{z\in[1,j-k+1]} \Theta(T_q[1,i], T_d[z,j])$.*

Solution 5 (DP-based Solution for cSimSub problem under LCSS measurement)**.** *In this solution, we only consider the situation that $k \geq 2$. Based on the state definition above, the state transition equation of cSimSub with LCSS similarity can be expressed as follows:*

$$S_{i,j,k} = \begin{cases} \max\{\mathbb{I}_{1,j}, S_{1,j-1,k-1}\} & if\ i=1, j\geq k \\ -\infty & if\ j<k \\ S_{i-1,j-1,k-1}+1 & if\ j\geq k, i>1, \mathbb{I}_{i,j}=1 \\ \max\{S_{i-1,j,k}, S_{i,j-1,k-1}\} & if\ j\geq k, i>1, \mathbb{I}_{i,j}=0 \end{cases}$$
$$where\ \mathbb{I}_{i,h}=1, if\ E(q_i,p_h)\leq\epsilon,\ otherwise\ \mathbb{I}_{i,h}=0 \tag{18}$$

When all elements of $S_{i,j,k}$ are completely computed (i.e., $k \in [1, C]$), the end index $j*$ of the most similar subtrajectory can be obtained through $j* = \arg\max_j S_{m,j,C}$. The start index $i*$ can be traced in a similar way to the solution of the SimSub.

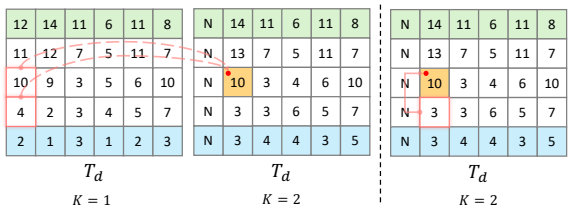

Fig. 2. A running example of cSimSub under the DTW measurement.

Proof. When $i = 1, j \geq k$,

$$\begin{aligned} S_{i,j,k} &= \max_{z\in[1,j-k+1]} LCSS(T_q[1,1], T_d[z,j]) \\ &= \max\{\max_{z\in[1,j-k]} LCSS(T_q[1,1], T_d[z,j]), \\ &\quad LCSS(T_q[1,1], T_d[j,j])\} \\ &= \max\{S_{1,j-1,k-1}, \mathbb{I}_{1,j}\} \end{aligned} \tag{19}$$

When $j < k$, the subtrajectory with length at least k is impossible ending with j. Thus, we set these values to infinity.

When $j \geq k, i > 1, \mathbb{I}_{i,j} = 1$,

$$\begin{aligned} S_{i,j,k} &= \max_{z\in[1,j-k+1]} LCSS(T_q[1,i], T_d[z,j]) \\ &= \max\{\max_{z\in[1,j-k]} LCSS(T_q[1,i], T_d[z,j]), \\ &\quad LCSS(T_q[1,i], T_d[j-k+1,j])\} \\ &= \max\{\max_{z\in[1,j-k]} LCSS(T_q[1,i-1], T_d[1,j-1])+1, \\ &\quad LCSS(T_q[1,i-1], T_d[j-k+1,j-1])+1\} \\ &= \max_{z\in[1,j-k]} LCSS(T_q[1,i-1], T_d[1,j-1])+1 \\ &= S_{i-1,j-1,k-1}+1 \end{aligned} \tag{20}$$

When $j \geq k, i > 1, \mathbb{I}_{i,j} = 0$,

$$\begin{aligned} S_{i,j,k} &= \max_{z\in[1,j-k+1]} LCSS(T_q[1,i], T_d[z,j]) \\ &= \max\{\max_{z\in[1,j-k]} LCSS(T_q[1,i], T_d[z,j]), \\ &\quad LCSS(T_q[1,i], T_d[j-k+1,j])\} \\ &= \max\{\max_{z\in[1,j-k]} \max\{LCSS(T_q[1,i-1], T_d[z,j]), \\ &\quad LCSS(T_q[1,i], T_d[z,j-1])\}, \max\{LCSS(T_q[1,i-1], \\ &\quad T_d[j-k+1,j]), LCSS(T_q[1,i], T_d[j-k+1,j-1])\}\} \\ &= \max_{z\in[1,j-k]} \max\{LCSS(T_q[1,i-1], T_d[z,j]), \\ &\quad LCSS(T_q[1,i], T_d[z,j-1])\} \\ &= \max\{S_{i-1,j,k}, S_{i,j-1,k-1}\} \end{aligned} \tag{21}$$

□

Pruning. According to the state definition of cSimSub, we have an observation: if the length of the most similar subtrajectory induced from $S_{i,j,k}$ is greater than $k+1$, the value of $S_{i,j,k+1}$ will equal $S_{i,j,k}$. Based on this observation, we develop a pruning algorithm to reduce the redundant computation when $k > 1$. Specifically, we utilize a matrix $P \in \mathcal{R}^{m\times n}$ to record the starting point for each $S_{i,j,k}$, i.e., $T_d[P[i,j],j]$ is the most similar subtrajectory ending with p_j, in which only one copy for the latest k exists. When DTW is used as the trajectory distance measurement, its calculation can be expressed as follows:

$$P_{i,j} = \begin{cases} j & if\ i=1 \\ 1 & if\ j=1 \\ P_{i,j-1} & if\ z=0 \\ P_{i-1,j-1} & if\ z=1 \\ P_{i-1,j} & if\ z=2 \end{cases} \tag{22}$$

Algorithm 2: Pruning-based Solution for cSimSub under DTW distance

Input: A query trajectory T_q with length m, a data trajectory T_d with length n, and constraint length C
Output: The most similar subtrajectory $T_d[i^*, j^*]$

1 **for** c *in* $[1, C]$ **do**
2 **if** c==*1* **then**
3 $S, P \leftarrow 0 \in \mathcal{R}^{m \times n}$;
4 **for** i *in* $[1, m]$ **do**
5 **for** j *in* $[1, n]$ **do**
6 update $S[i, j]$ according to Equation 5;
7 update $P[i, j]$ according to Equation 22;
8 **end**
9 **end**
10 $j^* \leftarrow \arg\min_j S[m, j]$;
11 $i^* \leftarrow P[m, j^*]$;
12 **if** $j^* - i^* + 1 \geq C$ **then**
13 break;
14 **end**
15 **end**
16 **else**
17 $S' \leftarrow 0 \in \mathcal{R}^{m \times n}$;
18 **for** i *in* $[1, m]$ **do**
19 **for** j *in* $[1, n]$ **do**
20 **if** $j - P[i, j] + 1 \geq c$ **then**
21 continue;
22 **end**
23 update $S'[i, j]$ according to Equation 15;
24 update $P[i, j]$ according to Equation 22;
25 **end**
26 **end**
27 $S \leftarrow S'$;
28 **end**
29 **end**
30 $j^* \leftarrow \arg\min_j S[m, j]$;
31 $i^* \leftarrow P[m, j^*]$;
32 **Return** $T_d[i^*, j^*]$;

where $z = \arg\min(S_{i,j-1,k}, S_{i-1,j-1,k}, S_{i-1,j,k})$ if $k = 1$, and $z = \arg\min(S_{i,j-1,k-1}, S_{i-1,j-1,k-1}, S_{i-1,j,k})$ when $k > 1$. By introducing P, we can quickly verify the length of the most similar subtrajectory, e.g., if $j - P_{i,j} + 1 \geq k$, there is no need to update the value of $S_{i,j,k}$ since the identified most similar subtrajectory ending with p_j already satisfies the length constraint. Apart from pruning during the element computation, we can utilize P to easily trace the most similar subtrajectory when the computation of all $S_{i,j,k}$ is completed. Specifically, we first obtain the ending index of the most similar subtrajectory $j* = \arg\min_j S_{m,j,C}$ and then read the starting index from P as $P[m, j*]$. It should be noted that this pruning method can also be adapted to other trajectory distance measurements, such as Frechet and EDR, without any modification.

Specifically, we elaborate our pruning algorithm for the cSimSub problem under DTW distance as shown in Algorithm 2, which takes a query trajectory T_q, a data trajectory T_d, and a user-specified minimal length of the searched subrajectory C as inputs. It loops from 1 to the length constraint C (lines 1). When current length c equals 1, we update S and P according to Equation 5 and Equation 22, respectively, for all elements (lines 2-9). Then, we will check the length of the searched most similar subtrajectory (lines 10-14). Specifically, we obtain the ending index of the most similar subtrajectory (line 10) and read the starting index from P (line 11). If the length of the most similar subtrajectory has already satisfied the user's requirement, the loop procedure will break (lines 12-14). Otherwise, we will continue to update the DP matrix until the length constraint is satisfied (lines 16-28). Specifically, S' representing the DP matrix with the length constraint c is firstly initialized (line 17). When looping at the element $[i, j]$, the length of the corresponding subtrajectory will be checked first (lines 20-22). If the length of the searched most similar subtrajectory ending with j is no less than c, the updating of S' and P will be ignored. Otherwise, the element of S' and P (i.e., $S'[i, j]$ and $S[i, j]$ represent $S_{i,j,k}$ and $S_{i,j,k-1}$ in Equation 15, respectively) will be updated. Finally, the most similar subtrajectory can be obtained when the loop procedure is finished (lines 31-32).

C. An Exact Solution of sSimSub

In this section, we provide a DP-based solution to solve the sSimSub problem to find a simplified trajectory that exhibits the highest similarity to itself. We empirically demonstrate that the self-similarity preserving is beneficial in improving the downstream query tasks, e.g., KNN query, as demonstrated in Section V-B2. Here, we introduce the state definition of DP to solve sSimSub as follows.

Definition 10 (State of DP for Solving sSimSub)**.** *Given a query trajectory T_q, a data trajectory T_d and a trajectory distance measurement $\Theta(\cdot, \cdot)$, $S_{i,j,k}$ denotes the state of DP used to solve the sSimSub problem, which represents the minimal distance between $T_q[1, i]$ and any subsequence of length k (k is a positive integer), starting at $T_d[1]$ and ending at $T_d[j]$, i.e., $S_{i,j,k} = \min_{1<p_{i_2}\cdots<p_{i_{k-1}}<j} \Theta(T_q[1, i], T_d^s < p_1, p_{i_2}, \cdots, p_{i_{k-1}}, p_j >)$.*

Solution 6 (DP-based Solution for the sSimSub problem with DTW distance)**.** *Since the sSimSub problem compulsively includes the first and last points in the compressed trajectory, we deal with the cases of $k = 1$, $k = 2$, and others, separately. Specifically, when $k = 1$ and $k = 2$, we have the following state transition equations.*

$$S_{i,j,k} = \begin{cases} \sum_{h=1}^{i} E(q_i, p_1) & if\ j = 1, k = 1 \\ +\infty & if\ j \neq 1, k = 1 \end{cases} \tag{23}$$

$$S_{i,j,k} = \begin{cases} E(q_i, p_j) + S_{i,1,k-1} & if\ i = 1, j \neq 1, k = 2 \\ +\infty & if\ j = 1, k = 2 \\ E(q_i, p_j) + \min\{S_{i,1,k-1}, S_{i-1,1,k-1}, S_{i-1,j,k}\} & otherwise \end{cases} \tag{24}$$

When $k > 2$, we have a universal state transition equations as follows.

$$S_{i,j,k} = \begin{cases} E(q_i, p_j) + \min\{\forall_{h=k-1}^{j-1} S_{i,h,k-1}\} & if\ i = 1, j \geq k, k > 2 \\ +\infty & if\ j < k, k > 2 \\ E(q_i, p_j) + \min\{S_{i-1,j,k}, v_{i,j,k}\} & otherwise \end{cases} \tag{25}$$

where $v_{i,j,k} = \min\{\forall_{h=k-1}^{j-1} S_{i,h,k-1}, \forall_{h=k-1}^{j-1} S_{i-1,h,k-1}\}$. When all elements of $S_{i,j,k}$ are completely computed (i.e.,

$k \in [1, C]$*), we trace the most similar subsequence, starting from* $S_{m,n,C}$ *to include the last point* p_n *in the returned subsequence, thereby satisfying the requirement of sSimSub.*

Proof. In the context of sSimSub, $k = 1$ implies simplifying a trajectory into a single point, which contradicts the requirement of sSimSub that the simplest trajectory should include the first and last points of the original trajectory. Thus, in this situation, we only initialize the elements in the first column, which will be used when $k = 2$. When $j = 1$ and $k = 1$,

$$\begin{aligned} S_{i,j,k} &= DTW(T_q[1,i], T_d[1,1]) \\ &= \sum_{h=1}^{i} E(q_i, p_1) \end{aligned} \tag{26}$$

When $j = 1, k = 2$, the subsequence ending with j fails to satisfy the length constraint. Thus, we directly initialize these elements with an infinite value. When $i = 1, j \neq 1, k = 2$, there is only one possible subsequence in $S_{i,j,k}$, i.e., a subsequence that starts with p_1 and ends with p_j, due to the constraint on the starting point of subsequence.

$$\begin{aligned} S_{i,j,k} &= DTW(T_q[1,1], T_d^s < p_1, p_j >) \\ &= E(q_1, p_j) + E(q_1, p_1) = E(q_i, p_j) + S_{i,1,k-1} \end{aligned} \tag{27}$$

For the other situation when $k = 2$,

$$\begin{aligned} S_{i,j,k} &= DTW(T_q[1,i], T_d^s < p_1, p_j >) \\ &= E(q_i, p_j) + \min\{DTW(T_q[1,i], T_d^s < p_1 >), \\ &\quad DTW(T_q[1,i-1], T_d^s < p_1 >), \\ &\quad DTW(T_q[1,i-1], T_d^s < p_1, p_j >)\} \\ &= E(q_i, p_j) + \min\{S_{i,1,k-1}, S_{i-1,1,k-1}, S_{i-1,j,k}\} \end{aligned} \tag{28}$$

When $j < k, k > 2$, the subsequence ending with j does not satisfy the length constraint. Thus, the corresponding $S_{i,j,k}$ is set to an infinity value. When $i = 1, j \geq k, k > 2$, we have

$$\begin{aligned} S_{i,j,k} &= \min_{1<p_{i_2}\cdots<p_{i_{k-1}}<j} DTW(T_q[1,1], T_d^s < p_1, p_{i_2}, \cdots, p_j >) \\ &= E(q_i, p_j) + \\ &\quad \min_{1<p_{i_2}\cdots<p_{i_{k-1}}<j} DTW(T_q[1,1], T_d^s < p_1, p_{i_2}, \cdots, p_{i_{k-1}} >) \\ &= E(q_i, p_j) + \min(\forall_{h=k-1}^{j-1} S_{i,h,k-1}) \end{aligned} \tag{29}$$

In other situations where $k > 2$, we have

$$\begin{aligned} S_{i,j,k} &= \min_{1<p_{i_2}\cdots<p_{i_{k-1}}<j} DTW(T_q[1,i], T_d^s < p_1, p_{i_2}, \cdots, p_j >) \\ &= \min_{1<p_{i_2}\cdots<p_{i_{k-1}}<j} E(q_i, p_j) + \min\{ \\ &\quad DTW(T_q[1,i-1], T_d < p1, \cdots, p_{i_{k-1}}, p_j >), \\ &\quad DTW(T_q[1,i], T_d < p1, \cdots, p_{i_{k-1}} >), \\ &\quad DTW(T_q[1,i-1], T_d < p1, \cdots, p_{i_{k-1}} >)\} \\ &= E(q_i, p_j) + \min\{S_{i-1,j,k}, \forall_{h=k-1}^{j-1} S_{i,h,k-1}, \\ &\quad \forall_{h=k-1}^{j-1} S_{i-1,h,k-1}\} \\ &= E(q_i, p_j) + \min\{S_{i-1,j,k}, v_{i,j,k}\} \end{aligned} \tag{30}$$

□

Example 2. *Figure 3 depicts a running example of the solution for the sSimSub problem under DTW measurement when* k *varies from* 1 *to* 3*. DTW defines three directions to transform to current state, i.e., left, left-down, and down. The left indicates that the previously matched pair must be* $T_q[i]$ *and* $T_d[j']$*, where* $1 \leq j' < j$ *rather than* $j' = j - 1$ *due to the property of subsequence (see Definition 3). Similarly, the left-down indicates that the previously matched pair must be* $T_q[i-1]$ *and* $T_d[j']$*, where* $1 \leq j' < j$*. For these two situations, the distance values have been obtained in previous state matrix* $S_{*,*,k-1}$*. The down indicates that the previously matched pair must be* $T_q[i-1]$ *and* $T_d[j]$*, which can also be obtained through the previous computation of current state matrix* $S_{*,*,k}$*. Thus, for the state* $S_{3,3,2}$ *in this example, we can obtained through the equation:* $S_{3,3,2} = E(T_q[3], T_d[3]) + \min\{S_{3,2,1}, S_{3,1,1}, S_{2,2,1}, S_{2,1,1}, S_{2,3,2}\} = 1 + \min\{Inf, 10, Inf, 4, 4\} = 5$.

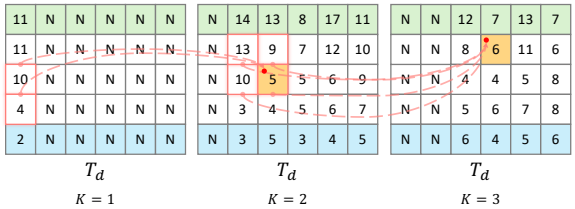

Fig. 3. A running example of sSimSub under the DTW measurement.

D. Time Complexity Analysis

For the SimSub problem, the time complexity of our solution is equivalent to that of the original trajectory distance measurement, as the only difference lies in the initialization of the DP matrix in the first row. Thus, we can achieve a time complexity of $O(mn)$ when well-known distance measurements, such as DTW, Frechet, EDR, and LCSS, are adopted.

In the case of the cSimSub problem, we extend the state of DP to a 3-dimensional matrix with dimensions $m \times n \times C$, where each element is scanned only once, and the computation of each element is in a constant time complexity, i.e., $O(1)$. Thus, this problem can be solved with a time complexity of $O(Cmn)$ at most. Furthermore, by applying the pruning technique, our algorithm for solving cSimSub can run even faster.

Regarding the sSimSub problem, except for the computation of $v_{i,j,k}$ (see Equation 25) that is an intermediate value tracking the previous minimum state value of DP based on the current state value, the remaining components exhibit the same time complexity as the algorithm for solving the cSimSub problem. Additionally, it is observed that $v_{i,j,k}$ can be recursively computed in time complexity of $O(1)$, since $v_{i,j,k} = \min\{v_{i,j-1,k-1}, S_{i,j-1,k-1}, S_{i-1,j-1,k-1}\}$, where $S_{i,j,k}$ denotes the current state value of DP. Thus, the total time complexity of our algorithm for solving the sSimSub problem is $O(Cmn)$.

Moreover, we also provide more comprehensive reviews of existing studies on the similar subtrajectory search problem as shown in Table II. In this table, we include a set of trajectory distance measurements, which can be computed through dynamic programming, such as STLCSS [36], [68], [69], and NetERP [50].

TABLE II
SUMMARY OF SIMILAR SUBTRAJECTORY SEARCH ALGORITHMS.

Algorithms	Accuracy	Constraint	Trajectory Distance Measurements							
			LCSS	LORS	DTW	Frechet	EDR	ERP	STLCSS	NetERP
Ours	**Exact**	**Yes**	$\boldsymbol{O(mn)}$	$\boldsymbol{O(mn)}$	$\boldsymbol{O(mn)}$	$\boldsymbol{O(mn)}$	$\boldsymbol{O(mn)}$	$\boldsymbol{O(mn)}$	$\boldsymbol{O(mn)}$	$\boldsymbol{O(mn)}$
CMA [12]	Exact	No	-	-	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$
ExactS [11]	Exact	Yes	$O(mn^2)$	$O(mn^2)$	$O(mn^2)$	$O(mn^2)$	$O(mn^2)$	$O(mn^2)$	$O(mn^2)$	$O(mn^2)$
Spring [66]	Exact	No	-	-	$O(mn)$	-	-	-	-	-
GB [67]	Exact	No	-	-	-	$O(mn)$	-	-	-	-
PSS [11]	Approx.	Yes	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$
POS [11]	Approx.	Yes	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$
POS-D [11]	Approx.	Yes	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$
RLS [11]	Approx.	Yes	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$
RLS-Skip [11]	Approx.	Yes	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$	$O(mn)$

V. EXPERIMENTS

A. Experimental Settings

Dataset. Our experiments are conducted on three real-world trajectory datasets, i.e., ChengDu[1], Porto[2], and Geolife[3]. ChengDu consists of around 1.2 million taxi trajectories located in Chengdu, China, from 2016-11-01 to 2016-11-07, with the average length around 82 released by DiDi Chuxing. Porto contains over 1.7 million trajectories collected from 2013 to 2014 in Porto, Portugal, with a sampling interval of 15s and the average length around 51. Geolife is collected from 182 users during a period of five years from 2007 to 2012 with the average length around 1,400 in Beijing.

Compared Methods. For the cSimSub problem, we compare our algorithms[4] with the following existing methods with slight modification to satisfy the length constraint.

- **ExactS [11].** Enumerating all possible subtrajectories and computes the similarities between the subtrajectories and query trajectory incrementally as much as possible, in which the subtrajectory whose length is less than C is eliminated.
- **SizeS [11].** Restricting the length of searched subtrajectories within the range $[m-\xi, m+\xi]$, where ξ is a user-specified hyperparameter, in which the subtrajectory whose length is less than C is ignored.
- **PSS, POS, and POS-D [11].** The splitting-based algorithms sequentially scan points in the data trajectory and find a splitting point to reduce the distance between searched subtrajectory and query trajectory, in which PSS considers both the prefix and suffix subtrajectories, POS and POS-D only consider the prefix subtrajectory. The splitting point will be ignored if the length of the resulting subtrajectory is less than C.
- **RLS**[5] **[11].** RLS is based on reinforcement learning and induced from the splitting-based algorithms, in which the splitting points are determined by the learned policy. We train a universal RLS model for all constraint lengths and constrain the searched process in the inference phase.
- **CMA**[6] **[12].** CMA is an exact method to solve SimSub. Specifically, it aims to find the optimal subtrajectory by computing the minimum cost of converting the query trajectory into the data trajectory, in which a set of costs, such as deletion, substitution, and insertion, for each trajectory measurement, are defined.

For the sSimSub problem, we compare our method with the following classical competitors.

- **Top-Down [22].** It repeatedly inserts a point with the largest error until the size of the simplified trajectory reaches the storage budget.
- **Bottom-Up [23].** It scans all points of the input trajectory and repeatedly drops the point with the smallest error until the number of remaining points is within the storage budget.
- **RLTS**[7] **[24].** It adopts the Bottom-Up strategy and drops points based on a learned policy instead of using heuristic rules.

Parameter Settings. For all competitors, we follow their original paper to set the hyper-parameters. Specifically, for SizeS, we set $\xi = 5$. For POS-D, we set the parameter D to 5. For RLS, we randomly sample 25K trajectory pairs for training and use a feedforward neural network with 2 layers, in which the ReLU activation with 20 neurons and Sigmoid activation with 2 neurons are used in the first and second layers, respectively. For RLTS, we randomly sample 1K trajectory pairs for training and use a feedforward neural network with 2 layers to implement the model architecture, in which the Tanh activation with 20 neurons and Softmax activation are used in the first and second layers, respectively.

Evaluation Metrics. For the cSimSub problem, following previous studies [11], we randomly sample 10K trajectory pairs from each dataset. In each pair, two trajectories are used as query and data, respectively. Then, the algorithm is performed to solve the cSimSub problem. We adopt three widely-used metrics, i.e., *Approximate Ratio (AR)*, *Mean Rank (MR)*, and *Relative Rank (RR)*, to evaluate the effectiveness [11], [12]. To specific, *AR* is defined as the ratio between the dissimilarity of the solution wrt a query trajectory, which is returned by an approximate algorithm and the optimal one. *MR*

[1]https://gaia.didichuxing.com

[2]http://www.geolink.pt/ecmlpkdd2015-challenge

[3]https://www.microsoft.com/en-us/research/publication/geolife-gps-trajectory-dataset-user-guide/

[4]https://github.com/LIWEIDENG0830/SimSub-DP

[5]https://github.com/zhengwang125/SimSub

[6]https://github.com/inabao/trajcSimilar

[7]https://github.com/zhengwang125/RLTS

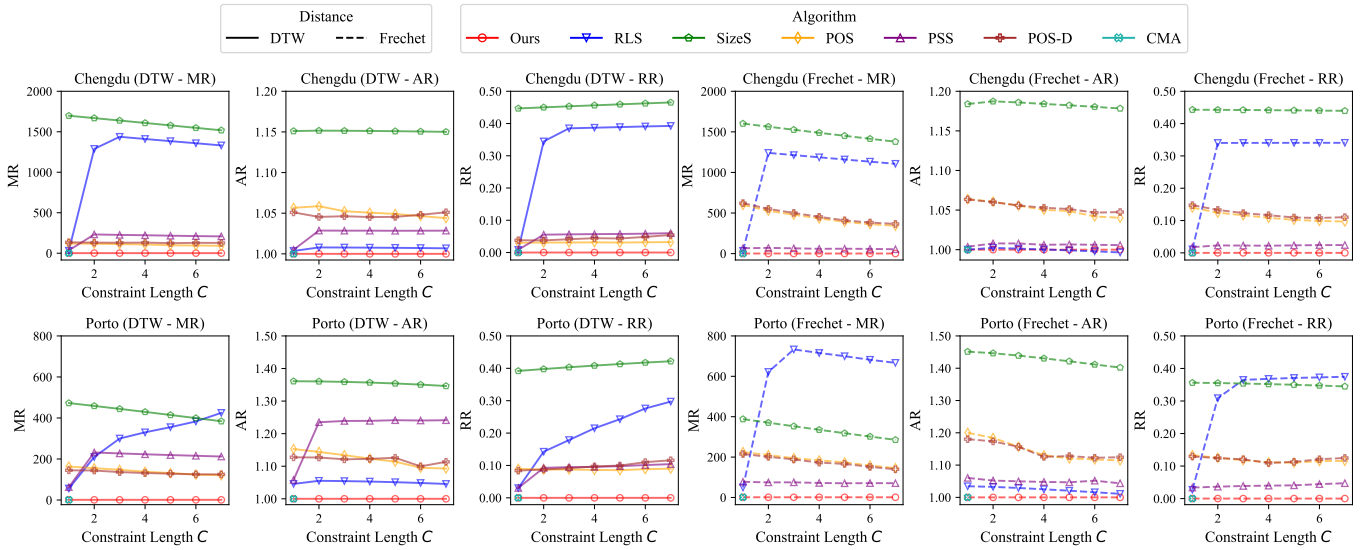

Fig. 4. Effectiveness of algorithms in terms of varying the constraint length.

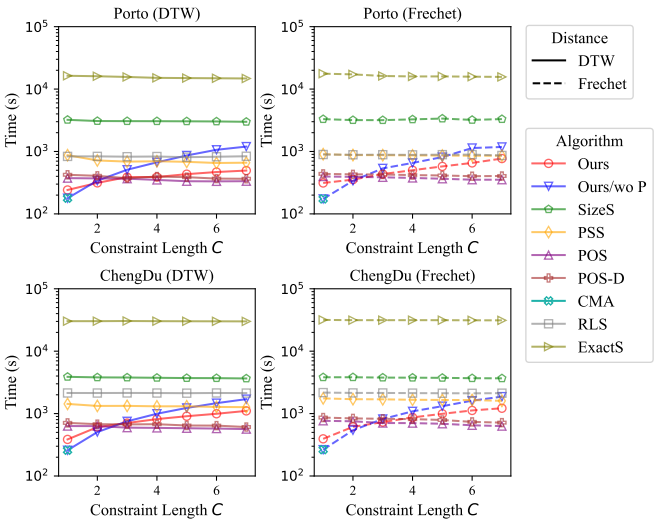

Fig. 5. Efficiency with varying length of constraint

is defined as the rank of the solution found by the algorithm, where all subtrajectories are sorted in ascending order of their dissimilarities wrt a query trajectory. *RR* is a normalized version of MR by the total number of subtrajectories of a data trajectory. A smaller MR or RR indicates a better algorithm. For the sSimSub problem, following the recent study [21], we perform KNN queries to study the query accuracy. Specifically, we randomly sample 11K trajectories from ChengDu (resp. 11K from Porto and 1.1K from Geolife), in which 1/11 of trajectories are used as query and the rest is as database. To evaluate the query performance, we use *accuracy (ACC)* for measuring the difference between query results in an original database D and those on a simplified database D'. We denote R_o and R_s as the trajectory sets returned by KNN queries from D and D', respectively. *ACC* can be calculated as $ACC = |R_o \cup R_s|/|R_o|$.

Evaluation Platform. We implement our methods and baselines in C++ and Python 3.7. The platform runs the Ubuntu 16.04 operating system with 48-cores Intel(R) CPU E5-2650 v4 @ 2.20GHz 256GB RAM.

B. Experimental Results

1) Effect of cSimSub: Firstly, we report the experimental results of cSimSub on ChengDu and Porto datasets for different algorithms with each distance function to evaluate the effectiveness of the proposed method, in which the constraint length C varies from 1 to 7 as shown in Figure 4. From these results, we can observe that compared with other approximation algorithms, RLS performs best when $C = 1$. For the other constraint values, the performance of RLS deteriorates sharply since it may need a special design to deal with the length constraint condition in the training process. Moreover, the performance of the other approximation algorithms is relatively stable with the varying of C, which demonstrates the robustness of these approaches. Compared with all the approximation competitors, both CMA and our algorithm show superior performance when $C = 1$ since they can provide the exact solution for SimSub under DTW and Frechet distance measurements. However, CMA does not work when $C \neq 1$, while our algorithm is flexible to fit users' different requirements in terms of the minimal length of the searched subtrajectory.

In addition, we also report the running time of all competitors as shown in Figure 5, in which ours/woP represents our algorithm without the proposed pruning algorithm. Compared with ExactS, the other algorithms achieve more than $10\times$ speed improvement. When the constraint length C equals 1, the efficiency of CMA and our algorithm is comparable, in which ours/woP achieves slightly faster than our original algorithm since the pruning technique only works when $C > 1$ and requires extra records and updates in terms of the matrix

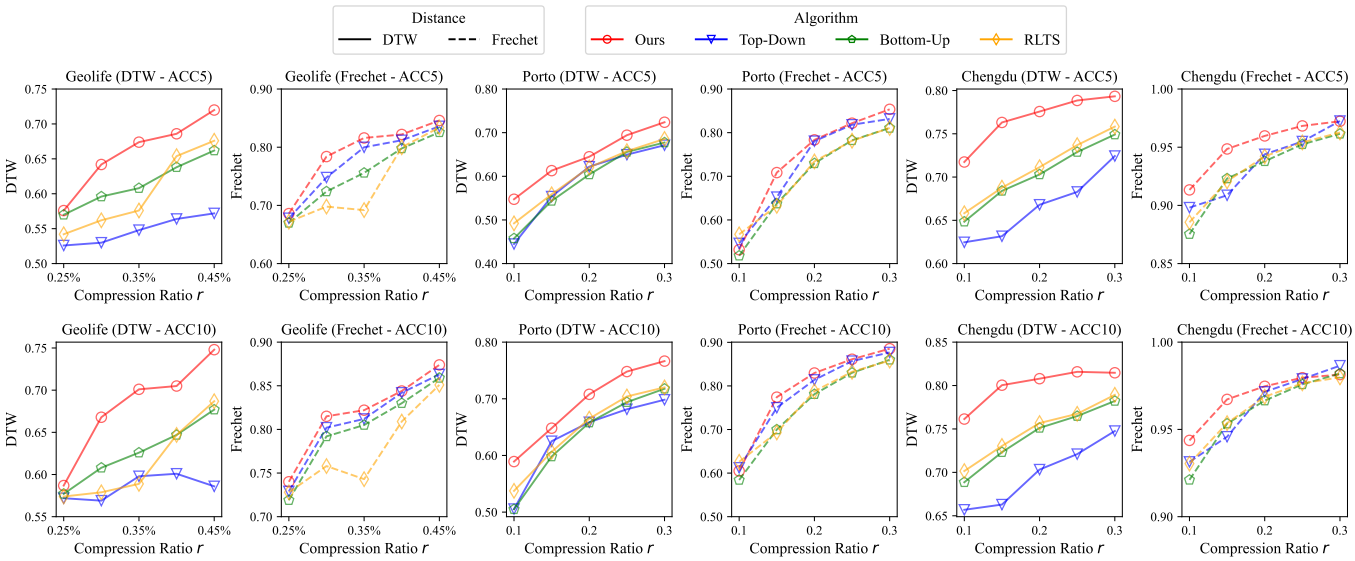

Fig. 6. Effectiveness of trajectory simplification algorithms in KNN query.

P. Compared with our algorithm without pruning, and with the increasing of C, the curve of our algorithm integrating the pruning technique is more flattened since it can reduce the number of updates of the DP matrix S. This observation demonstrates the effectiveness of our proposed pruning algorithm. Moreover, we can observe that the running time of other algorithms is stable when C varies from 1 to 7. However, we should notice that these algorithms are approximate, i.e., they have no guarantee in terms of the similarity of the searched subtrajectory. Despite the running time of our algorithm increases with the increase of constraint length C, its efficiency is still comparable with the approximation algorithms like PSS.

2) Effect of sSimSub: In this section, we study the effectiveness of the trajectory simplification algorithms in terms of the KNN query. The results on three datasets under different trajectory measurements are shown in Figure 6, in which K in KNN query is set to 5 and 10, denoted as ACC5 and ACC10, respectively. We can observe that compared with other competitors, the simplified trajectory through solving sSimSub with our algorithm can achieve the best query accuracy in most cases. This observation may indicate that self-similarity preserving is useful to maintain the trajectory distance distribution in the whole database. Moreover, our algorithm can achieve the largest improvement when DTW is adopted as the distance measurement. When Frechet distance is adopted, the competitors can also achieve good performance. This is because the Frechet distance is defined as the maximal distance among all matched points and the principle in these simplified trajectory algorithms, e.g., PED [55], [56], ensures that the removed points will not stray significantly from the original trajectory. Hence, the maximal distance among matched points between two trajectories remains relatively unaffected. In addition, we can observe that RLTS achieves a similar performance with Bottom-Up since RLTS adopts the Bottom-Up strategy with a learned policy to simplify trajectories.

VI. Conclusion

This paper explores the SimSub problem and extends its scope by introducing two variants, i.e., cSimSub and sSimSub. Specifically, cSimSub is a more general case of SimSub, in which users can control the minimal length of the most similar subtrajectory by specifying a positive integer C. We propose an algorithm based on dynamic programming to address the cSimSub problem with a time complexity of $O(Cmn)$. Our algorithm demonstrates adaptability to various trajectory distances, including DTW, Frechet, EDR, and LCSS, as long as dynamic programming can be implemented. Notably, from the solution equations, an interesting fact can be found that the only difference between our solution with the original distance computation lies in the the initialization of the first row of the DP matrix. As for the other variant, sSimSub, the goal is to find the most similar subsequence with length C from the trajectory itself, facilitating trajectory simplification. The empirical study shows that trajectory simplification through self-similarity preservation yields superior query accuracy in KNN queries compared with traditional trajectory simplification methods.

VII. Acknowledgement

This work is partially supported by NSFC (No. 62472068), Shenzhen Municipal Science and Technology R&D Funding Basic Research Program (JCYJ20210324133607021), and Municipal Government of Quzhou under Grant (No. 2023D044), and Key Laboratory of Data Intelligence and Cognitive Computing, Longhua District, Shenzhen.

REFERENCES

[1] H. Zhou, Y. Zhao, J. Fang, X. Chen, and K. Zeng, "Hybrid route recommendation with taxi and shared bicycles," *Distributed and Parallel Databases*, vol. 38, pp. 563–583, 2020.

[2] D. Yao, C. Zhang, Z. Zhu, J. Huang, and J. Bi, "Trajectory clustering via deep representation learning," *IJCNN*, pp. 3880–3887, 2017.

[3] P. K. Agarwal, K. Fox, K. Munagala, A. Nath, J. Pan, and E. Taylor, "Subtrajectory clustering: Models and algorithms," *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, 2018.

[4] K. Buchin, M. Buchin, J. Gudmundsson, M. Löffler, and J. Luo, "Detecting commuting patterns by clustering subtrajectories," *International Journal of Computational Geometry & Applications*, vol. 21, no. 03, pp. 253–282, 2011.

[5] A. Liang, B. Yao, B. Wang, Y. Liu, Z. Chen, J. Xie, and F. Li, "Subtrajectory clustering with deep reinforcement learning," *VLDBJ*, 2024.

[6] Y. Xu, J. Xu, J. Zhao, K. Zheng, A. Liu, L. Zhao, and X. Zhou, "Metaptp: An adaptive meta-optimized model for personalized spatial trajectory prediction," *KDD*, 2022.

[7] D. Yang, B. Fankhauser, P. Rosso, and P. Cudré-Mauroux, "Location prediction over sparse user mobility traces using rnns: Flashback in hidden states!" in *IJCAI*, 2020.

[8] D. Qiu, Y. Wang, Y. Zhao, L. Deng, and K. Zheng, "Citycross: Transferring attention-based knowledge for location-based advertising recommendation," in *2022 23rd IEEE International Conference on Mobile Data Management (MDM)*. IEEE, 2022, pp. 254–261.

[9] Y. Cui, H. Sun, Y. Zhao, H. Yin, and K. Zheng, "Sequential-knowledge-aware next poi recommendation: A meta-learning approach," *ACM Transactions on Information Systems (TOIS)*, vol. 40, no. 2, pp. 1–22, 2021.

[10] Y. Qin, Y. Fang, H. Luo, F. Zhao, and C. Wang, "Next point-of-interest recommendation with auto-correlation enhanced multi-modal transformer network," in *Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2022, pp. 2612–2616.

[11] Z. Wang, C. Long, G. Cong, and Y. Liu, "Efficient and effective similar subtrajectory search with deep reinforcement learning," *VLDB*, vol. 13, pp. 2312 – 2325, 2020.

[12] J. Jin, P. Cheng, L. Chen, X. Lin, and W. Zhang, "Efficient non-learning similar subtrajectory search," *VLDB*, vol. 16, no. 11, pp. 3111–3123, 2023.

[13] J.-G. Lee, J. Han, and K.-Y. Whang, "Trajectory clustering: a partition-and-group framework," in *SIGMOD*, 2007.

[14] P. Tampakis, C. Doulkeridis, N. Pelekis, and Y. Theodoridis, "Distributed subtrajectory join on massive datasets," *TSAS*, vol. 6, no. 2, pp. 1–29, 2020.

[15] D. Zhang, M. Ding, D. Yang, Y. Liu, J. Fan, and H. T. Shen, "Trajectory simplification: An experimental study and quality analysis," *VLDB*, vol. 11, pp. 934–946, 2018.

[16] X. Lin, S. Ma, J. Jiang, Y. Hou, and T. Wo, "Error bounded line simplification algorithms for trajectory compression: An experimental evaluation," *TODS*, vol. 46, pp. 1 – 44, 2021.

[17] Y. Zhao, S. Shang, Y. Wang, B. Zheng, Q. V. H. Nguyen, and K. Zheng, "Rest: A reference-based framework for spatio-temporal trajectory compression," *KDD*, 2018.

[18] X. Ding, L. Chen, Y. Gao, C. S. Jensen, and H. Bao, "Ultraman: A unified platform for big trajectory data management and analytics," *Proceedings of the VLDB Endowment*, vol. 11, no. 7, pp. 787–799, 2018.

[19] K. Zheng, Y. Zhao, D. Lian, B. Zheng, G. Liu, and X. Zhou, "Reference-based framework for spatio-temporal trajectory compression and query processing," *TKDE*, vol. 32, no. 11, pp. 2227–2240, 2019.

[20] Y. Zhao, S. Shang, Y. Wang, B. Zheng, Q. V. H. Nguyen, and K. Zheng, "Rest: A reference-based framework for spatio-temporal trajectory compression," in *SIGKDD*, 2018, pp. 2797–2806.

[21] Z. Wang, C. Long, G. Cong, and C. S. Jensen, "Collectively simplifying trajectories in a database: A query accuracy driven approach," *ICDE*, 2024.

[22] J. Hershberger and J. Snoeyink, "Speeding up the douglas-peucker line-simplification algorithm," 1992.

[23] E. J. Keogh, S. Chu, D. M. Hart, and M. J. Pazzani, "An online algorithm for segmenting time series," *ICDE*, pp. 289–296, 2001.

[24] Z. Wang, C. Long, and G. Cong, "Trajectory simplification with reinforcement learning," *ICDE*, pp. 684–695, 2021.

[25] L. Deng, H. Sun, Y. Zhao, S. Liu, and K. Zheng, "S2tul: A semi-supervised framework for trajectory-user linking," in *Proceedings of the sixteenth ACM international conference on web search and data mining*, 2023, pp. 375–383.

[26] L. Chen, Y. Gao, Z. Fang, X. Miao, C. S. Jensen, and C. Guo, "Real-time distributed co-movement pattern detection on streaming trajectories," *Proceedings of the VLDB Endowment*, vol. 12, no. 10, pp. 1208–1220, 2019.

[27] L. Chen, Y. Gao, X. Li, C. S. Jensen, and G. Chen, "Efficient metric indexing for similarity search and similarity joins," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 3, pp. 556–571, 2017.

[28] K. Zheng, Y. Zheng, N. J. Yuan, and S. Shang, "On discovery of gathering patterns from trajectories," in *2013 IEEE 29th international conference on data engineering (ICDE)*. IEEE, 2013, pp. 242–253.

[29] Y. Zhang, L. Deng, Y. Zhao, J. Chen, J. Xie, and K. Zheng, "Simidtr: Deep trajectory recovery with enhanced trajectory similarity," in *International Conference on Database Systems for Advanced Applications*. Springer, 2023, pp. 431–447.

[30] M. Chen, Y. Zhao, Y. Liu, X. Yu, and K. Zheng, "Modeling spatial trajectories with attribute representation learning," *TKDE*, vol. 34, no. 4, pp. 1902–1914, 2022.

[31] C. Wang, F. Zhao, H. Luo, Y. Fang, H. Zhang, and H. Xiong, "Towards effective transportation mode-aware trajectory recovery: Heterogeneity, personalization and efficiency," *IEEE Transactions on Mobile Computing*, 2024.

[32] Z. Liu, H. Miao, Y. Zhao, C. Liu, K. Zheng, and H. Li, "Lighttr: A lightweight framework for federated trajectory recovery," *arXiv preprint arXiv:2405.03409*, 2024.

[33] Y. Lun, H. Miao, J. Shen, R. Wang, X. Wang, and S. Wang, "Resisting tul attack: balancing data privacy and utility on trajectory via collaborative adversarial learning," *GeoInformatica*, vol. 28, no. 3, pp. 381–401, 2024.

[34] D. Hu, L. Chen, H. Fang, Z. Fang, T. Li, and Y. Gao, "Spatio-temporal trajectory similarity measures: A comprehensive survey and quantitative study," *TKDE*, 2023.

[35] B.-K. Yi, H. V. Jagadish, and C. Faloutsos, "Efficient retrieval of similar time sequences under time warping," *ICDE*, pp. 201–208, 1998.

[36] M. Vlachos, D. Gunopulos, and G. Kollios, "Discovering similar multidimensional trajectories," *ICDE*, pp. 673–684, 2002.

[37] L. Chen, M. T. Özsu, and V. Oria, "Robust and fast similarity search for moving object trajectories," in *SIGMOD*, 2005.

[38] H. Alt and M. Godau, "Computing the fréchet distance between two polygonal curves," *Int. J. Comput. Geom. Appl.*, vol. 5, pp. 75–91, 1995.

[39] P. Yang, H. Wang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "T3s: Effective representation learning for trajectory similarity computation," in *ICDE*. IEEE, 2021, pp. 2183–2188.

[40] L. Deng, Y. Zhao, J. Chen, S. Liu, Y. Xia, and K. Zheng, "Learning to hash for trajectory similarity computation and search," in *ICDE*. IEEE, 2024, pp. 4491–4503.

[41] L. Deng, Y. Zhao, Z. Fu, H. Sun, S. Liu, and K. Zheng, "Efficient trajectory similarity computation with contrastive learning," *CIKM*, 2022.

[42] Y. Chang, J. Qi, Y. Liang, and E. Tanin, "Contrastive trajectory similarity learning with dual-feature attention," in *ICDE*. IEEE, 2023, pp. 2933–2945.

[43] P. Yang, H. Wang, D. Lian, Y. Zhang, L. Qin, and W. Zhang, "Tmn: Trajectory matching networks for predicting similarity," in *ICDE*. IEEE, 2022, pp. 1700–1713.

[44] P. Han, J. Wang, D. Yao, S. Shang, and X. Zhang, "A graph-based approach for trajectory similarity computation in spatial networks," in *KDD*, 2021, pp. 556–564.

[45] H. Zhang, X. Zhang, Q. Jiang, B. Zheng, Z. Sun, W. Sun, and C. Wang, "Trajectory similarity learning with auxiliary supervision and optimal matching," in *IJCAI*, 2021, pp. 3209–3215.

[46] Z. Wang, C. Long, G. Cong, and C. Ju, "Effective and efficient sports play retrieval with deep representation learning," in *KDD*, 2019, pp. 499–509.

[47] X. Li, K. Zhao, G. Cong, C. S. Jensen, and W. Wei, "Deep representation learning for trajectory similarity computation," in *2018 IEEE 34th international conference on data engineering (ICDE)*. IEEE, 2018, pp. 617–628.

[48] D. Yao, G. Cong, C. Zhang, and J. Bi, "Computing trajectory similarity in linear time: A generic seed-guided neural metric learning approach," *ICDE*, pp. 1358–1369, 2019.

[49] Z. Fang, Y. Du, X. Zhu, D. Hu, L. Chen, Y. Gao, and C. S. Jensen, "Spatio-temporal trajectory similarity learning in road networks," in *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, 2022, pp. 347–356.

[50] S. Koide, C. Xiao, and Y. Ishikawa, "Fast subtrajectory similarity search in road networks under weighted edit distance constraints," *VLDB*, vol. 13, pp. 2188 – 2201, 2020.

[51] L. Deng, H. Sun, R. Sun, Y. Zhao, and H. Su, "Efficient and effective similar subtrajectory search: a spatial-aware comprehension approach," *TIST*, vol. 13, no. 3, pp. 1–22, 2022.

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

[53] M. Potamias, K. Patroumpas, and T. K. Sellis, "Sampling trajectory streams with spatiotemporal criteria," *SSDBM*, pp. 275–284, 2006.

[54] J. Muckell, J.-H. Hwang, V. Patil, C. T. Lawson, F. Ping, and S. S. Ravi, "Squish: an online approach for gps trajectory compression," in *International Conference and Exhibition on Computing for Geospatial Research & Application*, 2011.

[55] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, and R. Jurdak, "Bounded quadrant system: Error-bounded trajectory compression on the go," in *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 2015, pp. 987–998.

[56] N. Meratnia and R. A. de By, "Spatiotemporal compression techniques for moving point objects," in *EDBT*, 2004.

[57] B. Ke, J. Shao, Y. Zhang, D. Zhang, and Y. Yang, "An online approach for direction-based trajectory compression with error bound guarantee," in *APWeb*, 2016.

[58] B. Ke, J. Shao, and D. Zhang, "An efficient online approach for direction-preserving trajectory simplification with interval bounds," *MDM*, pp. 50–55, 2017.

[59] Z. Wang, C. Long, G. Cong, and Q. Zhang, "Error-bounded online trajectory simplification with multi-agent reinforcement learning," *KDD*, 2021.

[60] S. Wang, Z. Bao, J. S. Culpepper, Z. Xie, Q. Liu, and X. Qin, "Torch: A search engine for trajectory data," *SIGIR*, 2018.

[61] B. Zheng, L. Weng, X. Zhao, K. Zeng, X. Zhou, and C. S. Jensen, "Repose: Distributed top-k trajectory similarity search with local reference point tries," *ICDE*, pp. 708–719, 2021.

[62] T. Ichiye and M. Karplus, "Collective motions in proteins: A covariance analysis of atomic fluctuations in molecular dynamics and normal mode simulations," *Proteins: Structure*, vol. 11, 1991.

[63] J. K. Kearney and S. Hansen, "Stream editing for animation," 1990.

[64] M. T. Robinson, "The temporal development of collision cascades in the binary-collision approximation," *Nuclear Instruments and Methods in Physics Research Section B: Beam Interactions with Materials and Atoms*, vol. 48, no. 1-4, pp. 408–413, 1990.

[65] H. Su, S. Liu, B. Zheng, X. Zhou, and K. Zheng, "A survey of trajectory distance measures and performance evaluation," *The VLDB Journal*, vol. 29, pp. 3–32, 2020.

[66] Y. Sakurai, C. Faloutsos, and M. Yamamuro, "Stream monitoring under the time warping distance," *ICDE*, pp. 1046–1055, 2007.

[67] J. Gudmundsson, J. Pfeifer, and M. P. Seybold, "On practical nearest sub-trajectory queries under the fréchet distance," *ACM Transactions on Spatial Algorithms and Systems*, vol. 9, no. 2, pp. 1–24, 2023.

[68] T. Kahveci, A. K. Singh, and A. Gürel, "Similarity searching for multi-attribute sequences," *Proceedings 14th International Conference on Scientific and Statistical Database Management*, pp. 175–184, 2002.

[69] P. Patel, E. J. Keogh, J. Lin, and S. Lonardi, "Mining motifs in massive time series databases," *ICDM*, pp. 370–377, 2002.