Aspects of Spatial Trajectory Data Management–Compression and Clustering

Li, Tianyi

DOI (link to publication from Publisher):
10.54337/aau466212155

Publication date:
2021

Document Version
Publisher's PDF, also known as Version of record

ASPECTS OF SPATIAL TRAJECTORY DATA MANAGEMENT– COMPRESSION AND CLUSTERING

BY
TIANYI LI

DISSERTATION SUBMITTED 2021

AALBORG UNIVERSITY
DENMARK

Aspects of Spatial Trajectory Data Management– Compression and Clustering

Ph.D. Dissertation

Tianyi Li

Dissertation submitted November 1, 2021

Dissertation submitted: November 1, 2021

PhD supervisor: Prof. Christian S. Jensen
Aalborg University

PhD Co-supervisors: Prof. Torben Bach Pedersen
Aalborg University

Prof. Lu Chen
Zhejiang University

PhD committee: Associate Professor Gabriela Montoya (chair)
Aalborg University

Associate Professor Zhifeng Bao
RMIT University

Associate Professor Kyriakos Mouratidis
Singapore Management University

PhD Series: Technical Faculty of IT and Design, Aalborg University

Department: Department of Computer Science

ISSN (online): 2446-1628
ISBN (online): 978-87-7573-983-7

Published by:
Aalborg University Press
Kroghstræde 3
DK – 9220 Aalborg Ø
Phone: +45 99407140
aauf@forlag.aau.dk
forlag.aau.dk

Printed in Denmark by Rosendahls, 2021

Abstract

Large volumes of tracking data are being generated by GPS-enabled devices and subsequently stored in data warehouses. This data contains rich mobility-related information that may be utilized for discovering mobility patterns and other mobility characteristics that may in turn contribute to enabling a diverse range of services and applications such as aviation traffic monitoring and social networking. However, the considerable growth in the volumes of trajectory data presents difficulties in terms of efficient storage, and insufficient data quality precludes trajectories from effectively providing multiple services. As an example, in one specific setting, 2 billion trajectories are collected within a month by an application, and the GPS records have an average user range error of 7.8 meters with 95% probability.

The underlying characteristics of trajectory datasets enable the effective storage and analysis of trajectories. To be specific, trajectories or sub-trajectories may be similar to each other if they co-move in the same region during the same time of day. The resulting redundancy offers opportunities for storing the data more efficiently. Further, as a type of streaming data, movement trajectories represent gradual location changes, rather than abrupt changes. This may be beneficial for identifying trajectory patterns. The thesis provides methods that leverage these characteristics to enable the space efficient storage and high-quality pattern mining of trajectories. In particular, we focus on (i) trajectory compression and on (ii) trajectory clustering.

First, we propose a framework called UTCQ for uncertain trajectory compression and querying in road networks. We exploit the similarity between uncertain trajectory instances and provide a referential representation of trajectories, in order to achieve compact formats. As part of this, we propose a sample interval adaptive representation that compresses the temporal information of trajectories by taking into account variations in sample intervals. We also provide a reference selection algorithm based on a proposed Fine-grained Jaccard Distance to efficiently select trajectory instances as references. Next, a variable-length encoding scheme is presented for efficiently compressing referentially represented trajectories. Finally, we propose an index and develop filtering and validation techniques to support efficient queries over

compressed uncertain trajectories.

Second, we propose a framework called TRACE that enables compression, transmission, and querying of streaming trajectories in road networks in a fully online fashion. The framework employs a compact two-stage representation of streaming trajectories: a speed-based representation removes redundant information by employing vehicle speeds and accumulative distances, and a multiple-references based referential representation exploits subtrajectory similarities. Further, we adopt k-mer matching to adapt referential representation to online scenarios, and we extend it with reference selection, deletion, and rewriting functions that further improve compression. Next, we provide an efficient data transmission scheme for achieving low communication overhead. Finally, we propose indexing and filtering techniques to support efficient real-time range queries over compressed trajectories.

Third, we propose the notion of evolutionary clustering of streaming trajectories, abbreviated ECO, that enhances streaming-trajectory clustering quality via temporal smoothing that prevents abrupt changes in clusters across consecutive timestamps. Extending existing studies, we present new notions of snapshot and historical trajectory costs. We integrate these to formalize ECO and then formulate ECO as an optimization problem. Next, we prove that ECO can be solved approximately in linear time, which eliminates the iterative processes seen in previous studies. Further, we propose a minimal-group structure and a seed point shifting strategy to facilitate temporal smoothing. Finally, we present algorithms for each component of ECO along with optimization techniques.

We evaluate the proposed frameworks and methods by utilizing three real-life datasets and one synthetic dataset. The three real-life datasets are from two countries, specifically, Denmark and China, and three cities, specifically, Aalborg, Chengdu, and Hangzhou. The synthetic dataset contains 50 million trajectories that are divided into groups according to the similarities between each pair of trajectories. The experiments offer detailed insights into the efficiency and effectiveness of the proposed frameworks and methods. Specifically, UTCQ outperforms the state-of-the-art by a factor of more than two in terms of compression ratio and by more than one order of magnitude in terms of compression efficiency; TRACE improves the compression ratio by 67% and reduces the transmission cost by an order of magnitude; ECO outperforms the state-of-the-art in terms of both clustering quality and efficiency. In future research, it is of interest to study multiple-order compression to improve the compression ratio, to reduce the time delay and memory cost without compromising the compression ratio and transmission cost for real-time compression, and to utilize more information to improve clustering quality.

Resumé

Mobile objekter, såsom køretøjer og personer, med tilknyttede GPS-enheder genererer massive mængder af positionsdata, der beskriver objekternes bevægelser. Sådanne bevægelsesdata er potentielt nyttige i en bred vifte af anvendelser. De hastigt voksende mængder af data giver imidlertid udfordringer relateret til effektiv lagring. Desuden er lav datakvalitet en udfordring i forhold til mange anvendelser. Afhandlingen bidrager med kompressionsmetoder, der muliggør kompakt lagring af bevægelsesdata samt metoder til identifikation af klynger af mobile objekter.

Først beskriver afhandlingen et løsning, kaldet UTCQ, der muliggør komprimering af unøjagtige bevægelsesdata fra objekter, der bevæger sig i vejnetværk. Løsningen udnytter ligheder blandt de forskellige mulige bevægelser, som kan ligge til grund for de unøjagtige data, og repræsenterer mulige bevægelser som afvigelser fra udvalgte reference-bevægelser for at opnå et kompakt format. Dernæst beskrives hvordan sådanne repræsentationer kan afbildes til kompakte bitstrenge. Endelig beskriver afhandlingen teknikker til indicering og filtrering og validering, der muliggør effektive forespørgsler direkte på de komprimerede data.

For det andet beskriver afhandlingen en løsning, kaldet TRACE, der understøtter såkaldt online komprimering, hvor bevægelsesdata komprimeres i takt med, at de skabes og ankommer til systemet. Desuden muliggør TRACE, at data komprimeres der, hvor de ankommer, og så sendes i kompakt form til der, hvor de anvendes. Konkret tilpasses såkaldt k-mer matching til onlinescenarier, og teknikker til udvælgelse, sletning og omskrivning af reference-bevægelser introduceres med henblik på at opnå kompakte repræsentationer. Desuden præsenterer afhandlingen teknikker til effektiv datatransmission. Endelig beskriver den indicerings- og filtreringsteknikker, der understøtter effektive todimensionelle interval-forespørgsler direkte på komprimerede data.

For det tredje beskriver afhandlingen en løsning, kaldet ECO, der muliggør beregning af evolutionære klynger af mobile objekter i takt med at bevægelsesdata fra en population af objekter ankommer til systemet. For at forbedre kvaliteten af de beregnede klynger, så introducerer ECO tidsmæssige udjævningsteknikker, der forhindrer pludselige ændringer i klynger over kort

tid. Desuden defineres begreberne "snapshot omkostninger" og "historiske omkostninger", hvorefter de bruges til at formalisere det problem, som ECO løser. Dernæst formuleres problemet som et optimeringsproblem, og det bevises at problemet kan løses approksimativt i lineær tid. Dermed elimineres tidligere studiers omkostningstunge iterative processer.

Vi evaluerer de foreslåede løsninger empirisk ved brug af tre virkelige datasæt og et syntetisk datasæt. Evalueringerne giver detaljeret indsigt i løsningernes egenskaber og viser bl.a., at løsningerne er i stand til at levere bedre performance end de bedste eksisterende løsninger.

Acknowledgments

I would like to thank many people who have supported me during my Ph.D. studies.

First of all, I would like to express my sincere gratitude to my supervisor Prof. Christian S. Jensen. He has taught me so much; from writing skills to scientific research methods, which laid a solid foundation for my research. I also want to thank him for translating the Danish abstract of this thesis. I am very impressed by his meticulousness towards academic excellence. Further, his encouragement and support are great for my self-confidence. It is my pleasure to be his Ph.D. student.

Secondly, I would like to express my appreciation to my co-supervisor Prof. Torben Bach Pedersen. He has provided many unique ideas and insightful comments for my papers. He also provided writing tips, which significantly improved my papers. I learn a lot after every time I discuss with him. I'm very thankful for the time that he spent instructing me.

Thirdly, I am especially grateful to my co-supervisor Prof. Lu Chen. She provided guidance which allowed me to perform excellent research and helps me avoiding detours. She devoted so much time revising my drafts. I appreciate her dedication to my Ph.D project. Further, she provided a lot of support in my daily life. It's not an exaggeration to say that I would not have been able to complete this challenging journey without her guidance.

I would like to thank all my colleagues at the Database, Programming and Web Technologies group at Aalborg University for a fun filled and collegial working environment. The life of a Ph.D. student is fulled with adversities and solitude. It is a pleasure to have their accompany. Specifically, I would like to thank Asst. Prof. Jilin Hu and Asst. Prof. Tung Kieu, who helped me in many apsects in these past three years. I also thank the administrative staffs at AAU, especially Helle Westmark, Helle Schroll, and Ulla Øland who have made my Ph.D. life easier.

Lastly, I would like to express my gratitude for my husband who always stands by me and believes in me; my family who raised me and constantly encourages me; and my friends who play a positive role during my Ph.D. period.

Contents

Thesis Details

Thesis Title:	Aspects of Spatial Trajectory Data Management–Compression and Clustering
PhD Student:	Tianyi Li Aalborg University
PhD Supervisor:	Prof. Christian S. Jensen Aalborg University
PhD Co-supervisors:	Prof. Torben Bach Pedersen Aalborg University Prof. Lu Chen Zhejiang University

The main body of the thesis consists of the following papers.

(A) **Tianyi Li**, Ruikai Huang, Lu Chen, Christian S. Jensen, and Torben Bach Pedersen "*Compression of Uncertain Trajectories in Road Networks*," in PVLDB, pp. 1050–1063, 2020.

(B) **Tianyi Li**, Lu Chen, Christian S. Jensen, and Torben Bach Pedersen "*TRACE: Real-time Compression of Streaming Trajectories in Road Networks*," in PVLDB, pp. 1175–1187, 2021.

(C) **Tianyi Li**, Lu Chen, Christian S. Jensen, and Torben Bach Pedersen, and Jilin Hu "*Evolutionary Clustering of Streaming Trajectories*," (under review).

This thesis has been submitted for assessment in partial fulfillment of the Ph.D. degree. The thesis is based on the submitted or published scientific papers listed above. Parts of the content of the papers in the main body of the thesis are used directly or indirectly in the extended summary part of the thesis. As part of the assessment, co-author statements have been made available to the assessment committee and are also available at the Faculty. The permission for using the published and accepted articles in the thesis have been obtained from the corresponding publishers with the condition that they are cited and copyrights are placed prominently in the references.

Part I

Thesis Summary

Chapter 1

Introduction

1 Background and Motivation

Increased sophistication and deployment of computing, communication, and positioning technologies has resulted in the collection of rapidly growing volumes of mobility data [1]. For instance, Didi Chuxing ("DiDi"), a mobile platform and transportation services company, provides a variety of services to more than 550 million users all over the world[1]. DiDi's platform is used by tons of millions of drivers, car owners, and delivery service partners who earn flexible incomes. This results in more than 10 billion trajectories being generated a year[2]. As another example, Fitbit, a widely used type of wearable devices for activity tracking and fitness monitoring, collects trajectory data from its 23 million costumers at a high sampling rate[3]. Such trajectories encode us detailed mobility information that may be useful in services and applications in a variety of fields, e.g., traffic analysis [2], urban planing [3], mobility data mining [4], vehicular networks [5], and location-based social networks [6]. For example, identification of popular routes is beneficial for route recommendation.

However, the rapid growth in the amounts of available trajectory data leads to difficulties related to data storage. For example, if data is collected every two seconds, we need around 1GB of storage to store the data from just 800 objects per day. The size of data also incurs high costs of data transmission. Specifically, the expense of transmitting data over remote networks is excessively expensive, generally \$5 – \$7 per MB. Consequently, tracking 800 vehicles for a single day costs some \$5,000 to \$7,000 [7].

Further, although a trajectory can be conceptualized as a continuous time-

[1] https://www.didiglobal.com/science/brain
[2] https://www.businesswire.com
[3] http://expandedramblings.com/index.php/fitbit-statistics/

space function, it is collected by GPS-enabled devices as discrete sequences of timestamped locations in practice [8]. GPS receivers often do not record accurate data, due to sensor malfunctions, low signal strengths, interferences, etc. Statistics show that they have an average user range error of 7.8 meters with 95% probability [4]. Such errors may degrade the performance of location-based services significantly. For example, inaccurate locations may result in biased data prediction results and may compromise the training of models [9, 10].

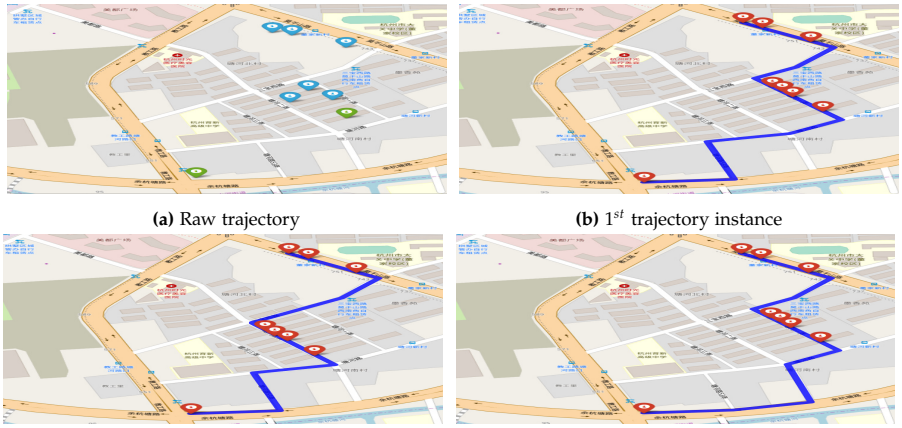
Motivated by the above considerations, we study two problems in the thesis: (i) trajectory compression that aims to mitigate the storage requirements and (ii) trajectory clustering that enables the mining of high-quality results by eliminating the adverse effect of noisy data.

1.1 Trajectory Compression

Trajectory compression aims to reduce the storage needed for storing trajectories and can be categorized into online compression and offline compression. Offline compression assumes that the complete history of collected trajectories is available. Usually, the best trade-off between compression quality and ratio can be achieved at the expense of high computational costs in the case of offline compression [11]. Online compression generally can only use the most recently received data, stored in a local buffer near the GPS devices to facilitate trajectory compression, and compressed trajectories are transferred to a central storage location in a streaming fashion in real-time.

Trajectory compression methods can also be classified into road network-based and simplification-based methods. Simplification-based methods compress raw trajectories that are made up by raw floating point value pairs (longitude and latitude). Such methods exclude less important trajectory points to achieve high compression ratios. However, simplification-based compression may then also reduce the utility of the data for analysis purposes. In contrast, road network-based compression is applied to trajectories that are projected onto a road network using map-matching algorithms [12, 13], called network-constrained trajectories [14]. A network-constrained trajectory can be represented in concise formats with little information loss. Thus, for certain types of data, typically data from network-constrained mobility, road network-based compression enables better compression with little or no reduction in quality [3].

Example 1.1
Figure 1.1a gives an example of a raw trajectory consisting of 8 GPS points recorded by a taxi in Hangzhou, China, while Figure 1.1b gives an example of a network-constrained trajectory, where each position in the raw trajectory in Figure 1.1a is mapped to a road-network location.



(a) Raw trajectory **(b)** 1^{st} trajectory instance

(c) 2^{nd} trajectory instance **(d)** 3^{rd} trajectory instance

Figure 1.1: Raw trajectory and network-constrained trajectories [14]

Motivated by the above analysis, we study road network-based trajectory compression, in both offline and online modes. In doing so, we take into account the following aspects:

- **Uncertainty of trajectories:** The uncertainty of trajectories is caused by two characteristics, a low sampling rate and inaccurate GPS positions [4, 15]. A low sampling rate can make multiple routes between two map-matched GPS positions possible, while position inaccuracy means that a raw GPS position may be map-matched to multiple road-network positions, e.g., by using probabilistic map-matching [12, 16].

  Example 1.2
  As shown in Figure 1.1a, all GPS points are off the road, and the sampling interval between two green points exceeds 4 minutes. Figures 1.1b, 1.1c, and 1.1d show trajectory instances generated from Figure 1.1a that are similar to each other.

  As suggested by Figure 1.1, probabilistic map-matching generally finds several potential road-network locations for a raw trajectory point, which generates a set of instances for a single uncertain trajectory and preserves the original information of raw trajectories as much as possible. However, storing multiple possible trajectory instances is costly, calling for effective trajectory compression schemes.

- **Effective storage reduction with high data usability**: Existing studies of network-based trajectory compression [17–22] have limited effect on storage reduction or achieve high compression at the cost of omitting useful information. First, most studies store auxiliary information, such

as frequent travel paths (FTP) and shortest travel paths (STP) [17–20] to compress trajectories, which increases the storage cost. Second, some studies discard useful information of trajectories, e.g., the timestamps and the exact locations of trajectories, for achieving high compression, reducing the data usability [21, 22].

- **Effective querying of compressed data**: Enabling effective querying is key to location-based services (LBS) and is a desirable property of compressed trajectories [22]. However, few studies take it into account [17, 20, 22]. Some studies [17, 20] need to perform full decompression before querying, which is very in-efficient. Although an existing study [22] proposes a partial decompression scheme for querying, the associated time cost is high because the proposed scheme is unable to support fast search on the temporal information of compressed trajectories.

- **Online compression without offline processing**: Existing proposals for online compression generally [23, 24] rely on offline training of prediction models using historical data. They discard data that can be predicted within a certain error bound during real-time compression. However, movement patterns on even the same road vary across time [25, 26], necessitating frequent re-training and incurring high transmission cost for delivering re-trained models, which limits the usability in practice.

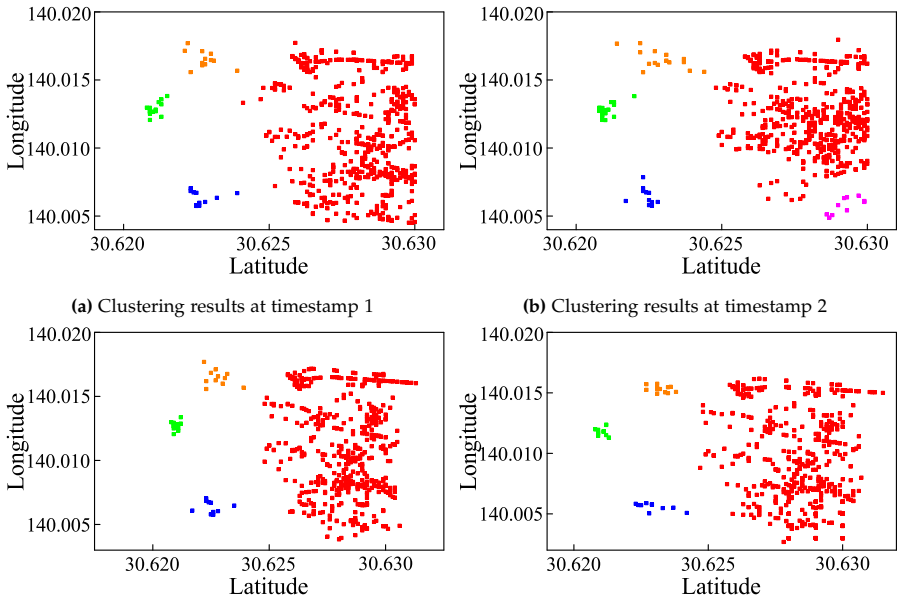
1.2 Trajectory Clustering

As a typical movement pattern discovery approach, trajectory clustering clusters similar trajectories to produce representative paths or common movement trends, which can serve as powerful tool to visualize mobility. Trajectory clustering is relevant to real-life applications, e.g., object motion prediction [27] and activity understanding [28]. For example, mining the common behaviors of hurricanes can facilitate forecasting the landfall of hurricanes; and in animal research, discovering common behaviors of animals may provide insight into the underlying causes of animal migration [29].

An important observation is that the cluster to which the trajectory of a moving object belongs may change over time, i.e., clustering result may evolve as time goes by. For example, traffic in road networks can be highly dynamic [29]. In order to consistently monitor the evolution of clusters and support real-time decision-making over moving objects, many studies propose different means of clustering trajectories in streaming settings [30–38].

Example 1.3
Figure 1.2 shows clustering results of trajectories from 717 taxis in Chengdu, China, over four timestamps. The interval between each two timestamps is

10s. Different clusters are plotted with different colors, i.e., four clusters exist at timestamp 1, 3, and 4, while five clusters exist at timestamp 2.



(a) Clustering results at timestamp 1

(b) Clustering results at timestamp 2

(c) Clustering results at timestamp 3

(d) Clustering results at timestamp 4

Figure 1.2: Evolving clustering results

Motivated by the above analysis, we study clustering of streaming trajectories. The main target of existing studies of streaming trajectory clustering is to efficiently update clusters continuously according to the most recent data. However, they are generally not robust to short-term fluctuations in the underlying trajectory data, which are caused by intermittent errors of GPS devices and/or unusual behaviors of moving objects [39].

Example 1.4
As many large cities are experiencing increasing traffic, road traffic management systems (RTMS) have been developed that reduce congestion by re-routing vehicles. Specifically, an RTMS re-routes vehicles from congested regions to less congested regions by analyzing data collected in real-time [40, 41]. Continuing Example 1.3 and assuming that the current timestamp is 2, an RTMS may determine to re-route vehicles from the space with the red cluster to the more open region between the red and the pink cluster, as the clustering result at timestamp 2 implies that few vehicles are located in this region. However, taking into account the smoothness of trajectory data and the clustering results at timestamps 1, 3, and 4, the clustering result at the timestamp 2 is likely to be wrong. In this case, the re-routing may be ineffective or even

lead to worse traffic. In addition, if the evolution of clusters in Figure 1.2 represents population migration, i.e., each cluster corresponds to a population, researchers will observe a suddenly emerged population (the pink cluster) at timestamp 2 and will find that it disappears at timestamp 3, which may be difficult to explain.

Example 1.4 indicates that it is beneficial to eliminate short-term fluctuations in clusters to achieve robustness to exceptional data. A naive approach is to perform cleaning before clustering. However, studies of two real-life datasets show that among the trajectories that cause mutations of clusters, 88.9% and 75.9% of the trajectories follow the speed constraint, while 97.8% and 96.1% of them are categorized as inliers [42]. Moreover, in real-time applications, it is impractical to correct previous clusters retroactively. Hence, it is difficult for existing cleaning techniques to facilitate smoothly shifting clustering sequences [43–45].

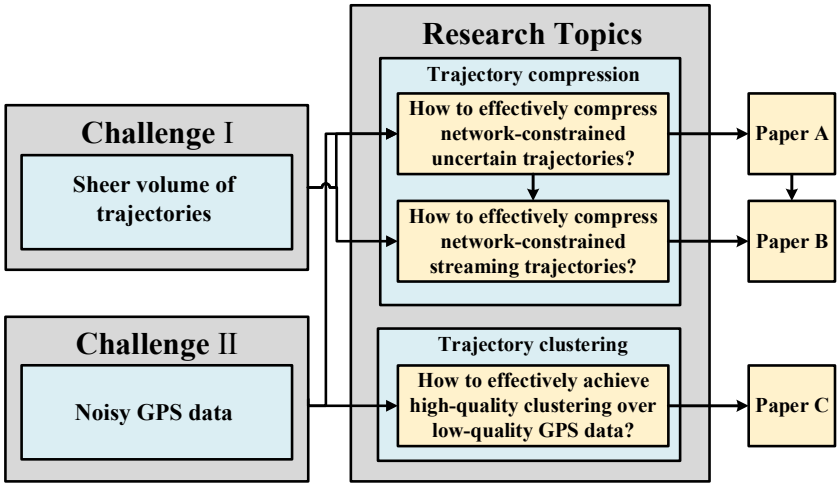


Figure 1.3: Thesis structure

2 Thesis structure

Figure 1.3 shows the thesis structure. As illustrated in Section 1, the thesis is motivated by two challenges: the sheer volume of trajectories and noisy GPS data. Trajectory compression is an effective strategy to reduce storage needs. As noisy GPS data yields uncertain trajectories, we first study compression of network-constrained uncertain trajectories in offline settings, which is the topic of Paper A. Then, we extend the study in Paper A to compress network-constrained trajectories in online settings, which is the topic of Paper B. Next, as low-quality GPS data degrades the performance of multiple pattern

discovery approaches of trajectories, we study how to eliminate its adverse effect on a typical pattern mining approach, namely clustering, which is the topic of Paper C.

Chapter 2 presents a framework for network-constrained uncertain trajectory compression and querying. Key aspects of the proposal include (i) a reference selection algorithm based on the notion of Fine-grained Jaccard Distance that is used to efficiently select trajectory instances as references; (ii) referential representation schemes for the different types of information contained in trajectories to achieve high compression ratios; and (iii) an index together with filtering techniques to support efficient queries over compressed uncertain trajectories.

Chapter 3 presents a framework that enables compression, transmission, and querying of network-constrained streaming trajectories in a fully online fashion. Specifically, the framework mainly encompasses (i) a compact two-stage representation of streaming trajectories, i.e., a speed-based representation that eliminates redundant information and a multiple-references based referential representation that exploits subtrajectory similarities; (ii) an online referential representation scheme extended with reference selection, deletion, and rewriting functions that further improves the compression performance; (iii) efficient data transmission scheme for achieving low transmission overhead; and (iv) an index and accompanying filtering techniques for supporting real-time range queries.

Trajectory clustering is presented in the Chapter 4. In that chapter, we define the concept of Evolutionary Clustering of streaming trajectOries (ECO) that improves streaming-trajectory clustering quality by means of temporal smoothing that prevents mutations in clusters across successive timestamps. By exploiting proposed notions of snapshot and historical trajectory costs, we formalize ECO and then formulate ECO as an optimization problem, and we prove that ECO can be performed approximately in linear time, thus eliminating the iterative processes employed in previous studies. In addition, we propose a minimal-group structure and a seed point shifting strategy to facilitate temporal smoothing. Finally, we present all algorithms supporting each components of ECO along with a set of optimization techniques. The recommended order to go through the thesis is *Paper A then Paper B then Paper C*.

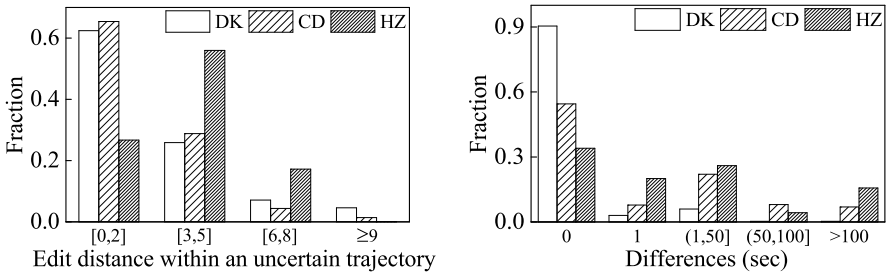
Chapter 2

Compression of Uncertain Trajectories in Road Networks

This chapter gives an overall introduction to Paper A [14]. It reuses content from that paper when this was considered most effective.

1 Problem Motivation and Statement

As illustrated in Section 1.1, trajectories can be uncertain due to the limitations of GPS-enabled devices. Probabilistic map-matching [46] has been developed to project raw GPS trajectories onto a road network. It generally finds multiple road-network locations for a raw trajectory point that preserve the original information of raw trajectories as much as possible. As a result, very large volumes of trajectory instances are generated for a single uncertain trajectory, calling for effective trajectory compression schemes.



(a) Similarity between two trajectory instances

(b) Sample intervals

Figure 2.1: Statistics of real-life datasets [14]

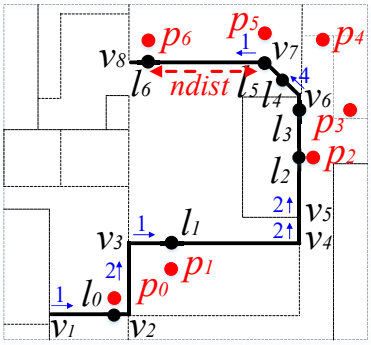
In order to effectively compress uncertain trajectories in road networks, two tasks must be accomplished. *The first task is achieving a high compression ratio.* Figure 2.1a, that shows statistics on three real-life datasets, Denmark (DK), Chengdu (CD), and Hangzhou (HZ), indicates the high similarity between trajectory instances. This motivates us to employ a referential representation for uncertain trajectories, which has been proven effective for compressing highly similar genome sequences [47–49]. *The second task is enabling efficient querying of compressed uncertain trajectories.* An existing study [22] develops an index for querying accurate compressed trajectories. However, the study does not consider the uncertainty of trajectories and is unsuitable for referentially represented trajectory instances. Thus, we design a novel indexing technique and associated query processing algorithms taking both into consideration.

We integrate the above contributions into a novel framework for Uncertain Trajectory Compression and Querying (UTCQ). First, we propose a two-stage representation, improved TED representation and referential representation. The improved TED representation removes more redundancy on the basis of the state-of-the-art TED model [22] and encompasses a novel SIAR scheme for representing temporal information. The referential representation exploits the similarities between trajectory instances to gain a more compact format on top of the improved TED representation. For achieving high performance referential representation, we select high quality references by using a proposed greedy reference selection algorithm. As part of this, we develop a Fine-grained Jaccard Distance (*FJD*) to efficiently measure the similarity between trajectory instances. Next, effective binary encoding schemes are presented for compressing represented uncertain trajectories. Finally, we devise algorithms to answer typical probabilistic queries over compressed uncertain trajectories. Specifically, an index structure with filtering and validation lemmas is proposed, which supports partial decompression and effective querying.

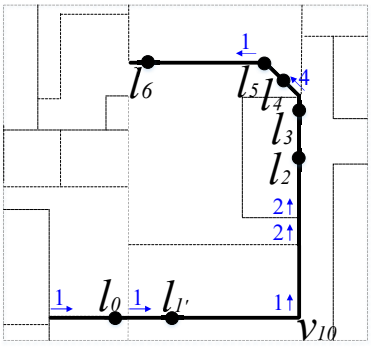
In summary, our main contributions are as follows:

- We propose a novel framework to compress uncertain trajectories that supports efficient probabilistic queries.
- We develop a representation that accommodates a novel encoding scheme for the temporal information of trajectories, and we use a referential representation to compress uncertain trajectories in road networks.
- We design an effective indexing structure and filtering techniques to facilitate query processing.
- We conduct extensive experimental evaluations on three real datasets to gain insight into the performance of the proposed framework.

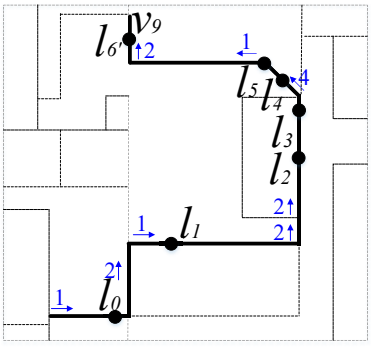
p_0=(120.0692, 30.28622, 5:03:25)
p_1=(120.0724, 30.28618, 5:07:25)
p_2=(120.0757, 30.28621, 5:11:26)
p_3=(120.0762, 30.28621, 5:15:26)
p_4=(120.0772, 30.28620, 5:19:25)
p_5=(120.0786, 30.28617, 5:23:25)
p_6=(120.0796, 30.28498, 5:27:25)



(a) Tu_1^1



(b) Tu_2^1



(c) Tu_3^1

Figure 2.2: Instances of the network-constrained uncertain trajectory Tu^1 [14]

2 Preliminaries

In this section, we formally define uncertain trajectories in road networks and introduce the TED representation [22]. The following definitions and examples are reproduced from [14].

2.1 Data Model

A **road network** is modeled as a directed graph $G = (V, E)$, where V is a set of vertices $v\,(x, y)$ denoting intersections or end points, and E is a set of directed edges $e\,(v_i \rightarrow v_j)$. Here, (x, y) denotes the 2D location of a vertex. For simplicity, we use v and $(v_i \rightarrow v_j)$ to denote a vertex and a directed edge of a road network, respectively.

Definition 2.1
A **mapped location** l is a network-constrained location in a road network G, represented as $\langle (v_s \rightarrow v_e), ndist, t \rangle$, where $ndist$ is the network distance between v_s and l on $(v_s \rightarrow v_e)$ and t is a timestamp.

We also denote a mapped location as $\langle (v_s \rightarrow v_e), ndist \rangle$ when the timestamp t is not considered. Given a mapped location $l = \langle (v_s \rightarrow v_e), ndist \rangle$, the *relative distance rd* of l w.r.t. $(v_s \rightarrow v_e)$ is the ratio of $ndist$ to the length of $(v_s \rightarrow v_e)$ (denoted as $|(v_s \rightarrow v_e)|$).

Definition 2.2
A **network-constrained uncertain trajectory** Tu^j contains a set of instances Tu^j_w $(1 \leq w \leq N^j)$ generated from a raw trajectory Tp. Each Tu^j_w is associated with a probability and is represented by a time-ordered sequence of mapped locations that is different from that of Tu^j_v $(1 \leq v \leq N^j \wedge v \neq w)$. All instances of Tu^j share the same temporal information for all mapped locations.

Example 2.1
In Figure 2.2a, $l_6 = \langle (v_7 \rightarrow v_8), ndist, 5{:}27{:}25 \rangle$ is the mapped location corresponding to p_6 and rd of $l_6 = \langle (v_7 \rightarrow v_8), ndist \rangle$ w.r.t. $(v_7 \rightarrow v_8)$ is $\frac{ndist}{|(v_7 \rightarrow v_8)|}$. Figure 2.2 shows a network-constrained uncertain trajectory Tu^1 generated from the raw trajectory $Tp = \langle p_0, ..., p_6 \rangle$. Tu^1 contains three instances, i.e., Tu^1_1, Tu^1_2, and Tu^1_3.

In the rest of the thesis, we use "trajectory" instead of "network-constrained trajectory" for simplification when this does not cause ambiguity.

2.2 TED representation

In the TED representation [22], an edge sequence is represented by a start vertex v followed by a sequence of outgoing edge numbers.

Definition 2.3
The **outgoing edge number** n_o (≥ 1) of an edge $(v_s \rightarrow v_e)$ means that $(v_s \rightarrow v_e)$ is the $n_o{}^{th}$ exit edge of v_s.

Example 2.2
In Figure 2.2, the edge sequence of Tu^1_1 is represented as $\langle 185190 \rightarrow$ 1,2,1,2,2,0,4, 1,0$\rangle$, 18590 is the ID of the start vertex the edge sequence, i.e., v_1.

TED uses a time flag bit-string to map timestamps to outgoing edge numbers.

Example 2.3
In Figure 2.2, the time flag bit-string of Tu^1_1 is $\langle$1,0,1,0,1,1,1,1,1$\rangle$, where the first 1 means that there is a mapped location on $(v_1 \rightarrow v_2)$ and the first 0 means that there is no mapped location on $(v_2 \rightarrow v_3)$.

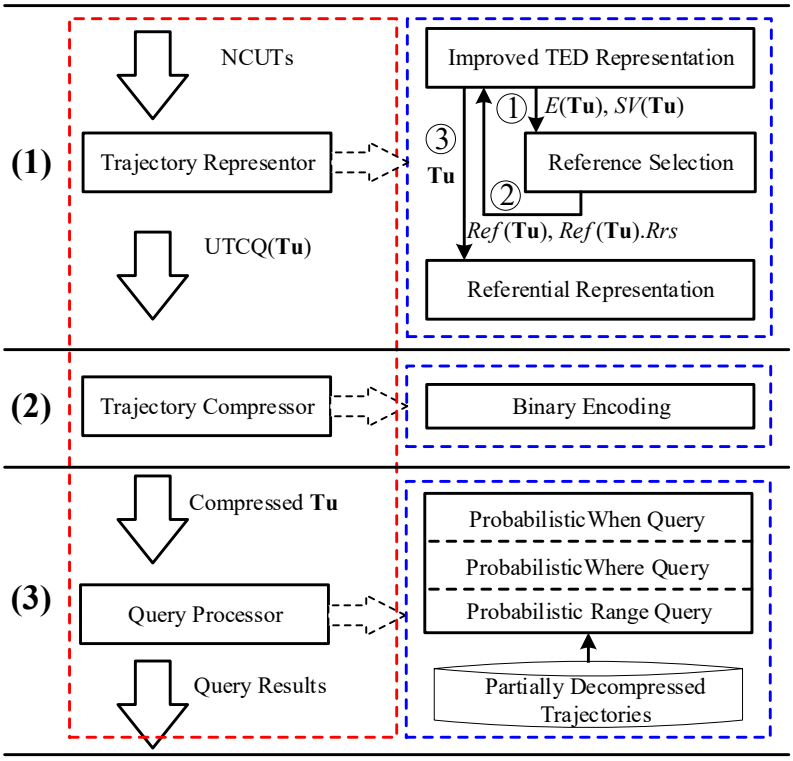


Figure 2.3: Framework [14]

3 UTCQ Framework

Figure 2.3 shows our framework for Uncertain Trajectory Compression and Querying (UTCQ) that contains three components, trajectory representor, trajectory compressor and query processor.

The ***trajectory representor*** transforms the network-constrained uncertain trajectories (NCUT) into a new compact format by improved TED representation, reference selection, and referential representation. Next, the ***trajectory compressor*** compresses the represented trajectories into binary codes. Finally, the ***query processor*** supports efficient querying of compressed trajectories.

4 Representation

This section covers the improve RED representation, referential representation, and reference selection. The following definitions and examples are reproduced from [14].

Table 2.1: Example of improved TED representation of Tu^1 in Figure 2.2 [14]

w	1	2	3
$SV(Tu^1_w)$	185190	185190	185190
$E(Tu^1_w)$	⟨1, 2, 1, 2, 2, 0, 4, 1, 0⟩	⟨1, 1, 1, 2, 2, 0, 4, 1, 0⟩	⟨1, 2, 1, 2, 2, 0, 4, 1, 2⟩
$D(Tu^1_w)$	⟨0.875, 0.25, 0.5, 0.875 0.5, 0, 0.875⟩	⟨0.875, 0.25, 0.5, 0.875 0.5, 0, 0.875⟩	⟨0.875, 0.25, 0.5, 0.875 0.5, 0, 0.5⟩
$T'(Tu^1_w)$	⟨0, 1, 0, 1, 1, 1, 1⟩	⟨1, 0, 0, 1, 1, 1, 1⟩	⟨0, 1, 0, 1, 1, 1, 1⟩
$Tu^1_w.p$	0.75	0.2	0.05

4.1 Improved TED representation

In the improved TED representation, we convert each uncertain trajectory instance Tu^j into a tuple $(SV(Tu^j_w), E(Tu^j_w), D(Tu^j_w), T'(Tu^j_w), Tu^j_w.p)$. $SV(Tu^j_w)$ is the start vertex ID of the first edge that is traversed by Tu^j_w. $D(Tu^j_w)$ and $Tu^j_w.p$ are the relative distance sequence and probability of Tu^j_w. $E(Tu^j_w)$ is the outgoing edge number sequence of Tu^j_w, excluding the start vertex. $T'(Tu^j_w)$ is the time flag bit-string of Tu^j_w. Note that the first and last bits 1 of $T'(Tu^j_w)$ are fixed according to [22] and thus are omitted to improve the compression.

Sample Interval Adaptive Representation (SIAR) of $T(Tu^j)$. Figure 2.1b reports the deviations between the actual sample intervals and the default ones of three real-life datasets. Since most of the actual sample intervals are very similar to their default ones, we record only the *difference* between them. Let the first timestamp of the time sequence of Tu^j be t_0 and T_s be the default sample interval. SIAR keeps t_0 as the first value in $T(Tu^j)$ and represents the following timestamps as $(t_{i+1} - t_i) - T_s$, where t_i is the i^{th} timestamp. The improved TED representation of Tu^1 is given in Table 2.1.

4.2 Referential Representation

Motivated by the high similarities among the trajectory instances representing a single uncertain trajectory (cf. Figure 2.1a), we select one or more instances as references for each uncertain trajectory. Then, other instances can be represented according to these references using a set of *factors* defined below.

Definition 4.1
Given a non-reference $Nref^j_{ik}$ and its corresponding reference Ref^j_i, $Nref^j_{ik}$ can be expressed as a list of **factors**, i.e., $Com_\phi(Nref^j_{ik}, Ref^j_i) = \langle \phi^j_{ik}(Ma_h) | 1 \leq h \leq H \rangle$, where H is the number of factors, and a factor $\phi^j_{ik}(Ma_h)$ denotes a subsequence in $Nref^j_{ik}$.

We use the *referential representation set* $Ref^j_i.Rrs$ to denote the set of $Nref^j_{ik}$ $(1 \leq k \leq |Ref^j_i.Rrs|)$ represented by Ref^j_i. Different components of trajecto-

Table 2.2: An referential representation example for Table 2.1 [14]

ϕ	$Com_\phi(Nref_{11}^1, Ref_1^1)$	$Com_\phi(Nref_{12}^1, Ref_1^1)$
SV	$\emptyset$	$\emptyset$
E	$\langle(0, 1, 1), (2, 7)\rangle$	$\langle(0, 8, 2)\rangle$
D	$\emptyset$	$\langle(6, 0.5)\rangle$
T'	$\langle(1, 2), (3, 4)\rangle$	$\emptyset$

ries are referentially represented by different formats of factors. We adopt the format (S, L, M) to encode $E_{ik}^j(Ma_h)$ and the format (S, L) to encode $T'^j_{ik}(Ma_h)$ [49], where S is the *start position* of the subsequence in the reference, L is the *length* of the subsequence, and M is the first *mis-matched element* following the subsequence. $T'^j_{ik}(Ma_h)$ omits M because it can be inferred easily by the reference. $D_{ik}^j(Ma_h)$ is represent using the format (pos, rd), where pos is the position of the different value rd. The referential representation of Tu^1 is given in Table 2.2.

4.3 Reference Selection

As can be seen from Table 2.2, the more similar a reference and a non-reference are, the more redundancy can be removed and thus the higher compression ratio can be achieved. Intuitively, trivially computing the similarity between each pair of trajectories is time-consuming. Thus, inspired by a previous study [49], we propose to use the similarity between the referential representations of trajectory instances to approximate their exact one. This requires us to select a pivot for each uncertain trajectory for referentially representing other trajectory instances. The pivot selection generally follows that of an an existing study [50]. Thereafter, we approximately estimate the similarity between two represented edges sequences $E(Tu_w^j)$ to $E(Tu_v^j)$ against a pivot piv_i by a newly defined *Fine-grained Jaccard Distance* (*FJD*),

$$FJD(Tu_w^j \rightarrow Tu_v^j, piv_i)(w \neq v) = \frac{\sum_{h'=1}^{H'} sim(E_{iv}^j(Ma_{h'}), Com_E(Tu_w^j, piv_i))}{\max\{H, H'\}}, \quad (2.1)$$

where H and H' denote the number of factors in $Com_E(Tu_w^j, piv_i)$ and $Com_E(Tu_v^j, piv_i)$, respectively, while $E_{iv}^j(Ma_{h'})$ denotes the h'^{th} factor $(S_{h'}^{iv}, L_{h'}^{iv})$ in $Com_E(Tu_v^j, piv_i)$. In addition, we use $sim(E_{iv}^j(Ma_{h'}), Com_E(Tu_w^j, piv_i))$ to measure the similarity between $E_{iv}^j(Ma_{h'})$ and $Com_E(Tu_w^j, piv_i)$ as follows.

$$sim(E_{iv}^j(Ma_{h'}), Com_E(Tu_w^j, piv_i)) = \frac{\max_{h=1}^{H}(E_{iw}^j(Ma_h) \cap E_{iv}^j(Ma_{h'}))}{\max\{L_{\max}^{iw}, L_{h'}^{iv}\}} \quad (2.2)$$

We define $E^j_{iw}(Ma_h) \cap E^j_{iv}(Ma_{h'})$ as $\max\{\min\{S^{iw}_h + L^{iw}_h, S^{iv}_{h'} + L^{iv}_{h'}\} - \max\{S^{iw}_h, S^{iv}_{h'}\}, 0\}$, and $L^{iw}_{\max} = \arg\max_{L^{iw}_h} E^j_{iw}(Ma_h) \cap E^j_{iv}(Ma_{h'})$. Example 4.1 gives the process of *FJD* computation.

Example 4.1
Consider the example in Table 2.2. Given $piv_1 = Tu^1_3$, we have $Com_E(Tu^1_1, piv_1) = \langle (0,8), (5,1) \rangle$ and $Com_E(Tu^1_2, piv_1) = \langle (0,1), (0,1), (2,6), (5,1) \rangle$. Next we compute $\frac{E^1_{11}(Ma_1) \cap E^1_{12}(Ma_1)}{\max\{L^{11}_1, L^{12}_1\}} = \frac{1}{8}$ for getting $sim(E^1_{12}(Ma_1), Com_E(Tu^1_1, piv_1))$. Similarly, we are able to gain $sim(E^1_{12}(Ma_2), Com_E(Tu^1_1, piv_1)) = \frac{1}{8}$, $sim(E^1_{12}(Ma_3), Com_E(Tu^1_1, piv_1)) = \frac{3}{4}$, and $sim(E^1_{12}(Ma_4), Com_E(Tu^1_1, piv_1)) = 1$. Therefore, $FJD(Tu^1_1 \rightarrow Tu^1_2, piv_1) = (\frac{1}{8} + \frac{1}{8} + \frac{3}{4} + 1)/4 = \frac{1}{2}$.

Next, we define a score function $SF(Tu^j_w, Tu^j_v) = Tu^j_w.p \cdot \max_{i=1}^{n^j_p} FJD(Tu^j_w \rightarrow Tu^j_v, piv_i)$ $(w \neq v)$ that evaluates the performance of representing Tu^j_v by Tu^j_w. This way, the optimal reference for Tu^j_v can be derived by:

$$Ref(Tu^j_v) = \arg\max_{Tu^j_w} SF(Tu^j_w, Tu^j_v) \tag{2.3}$$

Finally, we propose a greedy algorithm for reference selection of uncertain trajectories. We first build a score matrix **SM** according to *SF*, where $SM[w][v] = SF(Tu^j_w, Tu^j_v)$. Then, we always select Tu^j_w as a reference such that $SM[w][v]$ $(1 \leq v \leq |Tu^j|)$ is the current maximal element in **SM** and remove it from **SM**. This process continues until **SM** is empty. Example 4.2 gives the reference selection of Tu^1.

Example 4.2
Assuming that we only select Tu^1_3 as a pivot for Tu^1, we get an **SM**. Then we find the maximum in **SM**, i.e., $SF(Tu^1_1, Tu^1_2)$, based on which we get a reference Tu^1_1 and add Tu^1_2 to its *Rrs*. Next, $SM[w'][2] \cup SM[2][w''] \cup SM[v'][1]$ $(1 \leq w', w'', v' \leq 3)$ are removed from **SM** as (i) Tu^1_1 has been assigned as a reference and (ii) Tu^2_1's reference has been determined, i.e., Tu^1_1. This process is shown below,

$$\mathbf{SM} = \begin{bmatrix} 0 & \frac{3}{8} & \frac{1}{3} \\ \frac{7}{80} & 0 & \frac{1}{30} \\ \frac{1}{40} & \frac{1}{80} & 0 \end{bmatrix} \rightarrow \begin{bmatrix} \cancel{0} & \cancel{\frac{3}{8}} & \frac{1}{3} \\ \cancel{\frac{7}{80}} & \cancel{0} & \cancel{\frac{1}{30}} \\ \cancel{\frac{1}{40}} & \cancel{\frac{1}{80}} & 0 \end{bmatrix}$$

Then we add Tu^1_3 to $Tu^1_1.Rrs$ due to $SM[1][3] > SM[3][3]$, and remove $SM[w'][3] \cup SM[3][w'']$ $(1 \leq w', w'' \leq 3)$ from **SM**. Finally, since $\mathbf{SM} = \emptyset$, we return the reference Tu^1_1 with its $Rrs = \{Tu^1_2, Tu^1_3\}$ for Tu^1.

5 Compression

We adopt the PDDP-tree [22] to encode $D(Ref)$ and $Ref.p$ that are floats. We use the error bounds η_D and η_p to constrain their compression accuracy. Given the maximum outgoing number of a road network, o, each outgoing edge number in $E(Ref)$ is encoded with fixed length, i.e., $\lceil \log_2 o \rceil$.

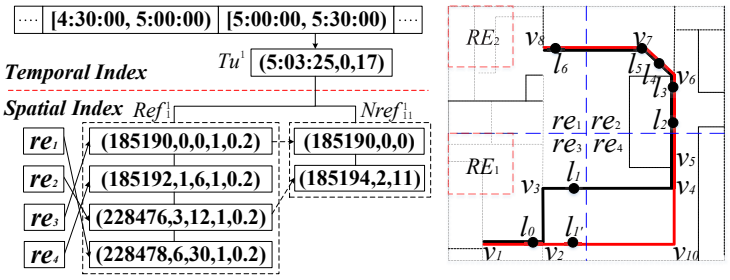
Improved Exp-Golomb Encoding. Exp-Golomb encoding, that is well-known for encoding positives, encodes smaller values with shorter lengths and encodes larger values with longer lengths. Since we find that small deviations between the actual sample intervals and the default ones occur much more frequently than large ones (cf. Figure 2.1b), we adopt the Exp-Golomb encoding [51] to encode $\triangle t_i = (t_{i+1} - t_i) - T_s$ $(0 \leq i < |T(Tu^j)| - 1)$ (cf. Section 4). However, as $\triangle t_i$ may be negative, the Exp-Golomb encoding needs to be modified.

Assuming that the longest actual sample interval is T_l, we have $\triangle t_i \in (-T_s, T_l - T_s]$ $(0 \leq i < |T(Tu^j)| - 1)$. We divide $[0, \max\{T_s - 1, T_l - T_s\}]$ into n groups, where $n = \lceil \log_2(\max\{T_s - 1,\ T_l - T_s\} + 1) \rceil$, and the range of the $j^{th} (j \geq 0)$ group is $\left[-2^{j+1} + 2,\ \ -2^j + 1\right] \cup \left[2^j - 1, 2^{j+1} - 2\right]$. This way, all the possible deviations between the actual sample interval and the default one can be covered by $[-\max\{T_s - 1, T_l - T_s\}, \max\{T_s - 1, T_l - T_s\}] \subseteq [-2^n + 2, 2^n - 2]$. The offset of $\triangle t_i$ in the j^{th} group is given by $|\triangle t_i| - (2^j - 1)$. Moreover, we add one 1 bit immediately before the offset if $\triangle t_i$ is a negative digit; otherwise, 0 is added.

Variable-Length Encoding. Let $|E(Ref_i^j)|$ be the length of $E(Ref_i^j)$ and o be the maximum number of outgoing edges for any vertex $v \in V$. During the variable-length encoding of a factor (S, L, M) in $E(Nref_{ik}^j)$, S takes $\lceil \log_2 |E(Ref_i^j)| + 1 \rceil$ bits, L takes $\lceil \log_2 |E(Ref_i^j)| \rceil$ bits, and M takes $\lceil \log_2 o \rceil$ bits. Similarly, when encoding a factor of (S, L, M) in $T'(Nref_{ik}^j)$, both S and L take $\log_2 \lceil |T'(Ref_i^j)| \rceil$ bits, while M takes 1 bit. Further, *pos* in each factor of $D(Nref_{ik}^j)$ takes $\lceil \log_2 |D(Ref_{ik}^j)| \rceil$ bits, and *rd* is encoded by a PDDP-tree [22]. This way, binary codes of different non-references may have different lengths depending on their similarities to the corresponding references, which further improve compression.

6 Query Processing

This section introduces an index structure and filtering and validating techniques for probabilistic queries. The following definitions and examples are reproduced from [14].



(a) Index Structure **(b)** Tu_1^1 and Tu_2^1

Figure 2.4: StIU built on $\boldsymbol{Tu} = \{Tu^1\}$ depicted in Figure 2.1 [14]

6.1 StIU Index

We propose an index, called Spatio-temporal Information based Uncertain Trajectory Index (StIU), to support efficient probabilistic queries by partial decompression [22]. The index contains two parts, as shown in Figure 2.4a. The upper part indexes the temporal information of trajectories, while the lower part supports effective spatial search.

Temporal Index of StIU. We partition the time line into equal-length time intervals. The information of an uncertain trajectory Tu^j corresponding to a time interval is stored in a tuple ($t.start$, $t.no$, $t.pos$), where $t.start$ is the earliest timestamp of Tu^j falling into the time interval, $t.no$ indicates that $t.start$ is the $t.no^{th}$ timestamp in $T(Tu^j)$, and $t.pos$ refers to the matching position of the $(t.no+1)^{th}$ timestamp in $\hat{T}(Tu^j)$.

Spatial Index of StIU. We define the concept of *final vertex*, before introducing the spatial index.

Definition 6.1
A **final vertex** of a trajectory instance Tu_w^j w.r.t. a region re is a vertex in G that is traversed by the trajectory instance immediately before reaching re, denoted as $Tu_w^j.fv$ of re.

For a reference, the tuple corresponding to re has the form ($fv.id$, $fv.no$, $d.pos$, p_{total}, $p_{\max}$), where 1) $fv.id$ (≥ 0) is the ID of $Ref_i^j.fv$ of re; 2) $fv.no$ indicates the position of $fv.id$ in $E(Ref_i^j)$; 3) $d.pos$ is the matching position of the $d.no^{th}$ relative distance in $\hat{D}(Ref_i^j)$ such that it is the last relative distance of Ref_i^j before entering re; 4) with Ω defined as the subset of all trajectory instances $Ref_i^j \cup Ref_i^j.Rrs$ that overlap re, p_{total} is then the sum of the probabilities of all instances in Ω; and 5) $p_{\max} = Nref_{ik'}^j.p$ such that $\forall Nref_{ik}^j \in \Omega$: $Nref_{ik'}^j.p \geq Nref_{ik}^j.p$. If $\forall Nref_{ik}^j \in Ref_i^j.Rrs$, $Nref_{ik}^j$ does not overlap re, $p_{\max}$ is set to 0. For a non-reference, the tuple corresponding to re has the form $(fv.id, p_{total}, p_{\max})$, where p_{total} and $p_{\max}$ are the same as those for the reference. The binary code

of a sequence *seq* is denoted as $\hat{seq}$.

Example 6.1
Figure 2.4a shows an StIU index built on Tu^1, where Tu_1^1 (used as Ref_1^1) and Tu_2^1 (used as $Nref_{11}^1$) are instances of the uncertain trajectory Tu^1 in Figure 2.2, where the IDs of v_1, v_2, v_3, v_4, v_5, and v_7 are 185190, 185191, 185192, 185194, 228476, and 228478, respectively; the mapping positions of the relative distances of l_1, l_2, and l_5 in $\hat{D}(Ref_1^1)$ are 6, 12, and 30, respectively; and the maximum outgoing edge number of the road network in Figure 2.4b is 7. If we locate a query result in re_2, we can obtain the mapping position of v_5 in $\hat{E}(Ref_1^1)$ and the relative distance w.r.t l_2 on (v_5, v_6) in $\hat{D}(Ref_1^1)$ by the tuple (228476, 3, 12, 1, 0.2). This way, we get the information of Ref_1^1 in re_2 without decompressing from the beginning of those sequences.

6.2 Probabilistic Queries

Based on StIU, three representative types of queries, namely probabilistic where, when, and range queries, can be performed.

Definition 6.2
Given a timestamp t, a probability α, and a compressed trajectory stream Tr^n, a **probabilistic where query where**(Tu^j, t, α) returns the set of mapped locations at time t of the instances $Tu_w^j \in Tu^j$ with $Tu_w^j.p \geq \alpha$. Each location is given as $\langle (v_s \rightarrow v_e), ndist \rangle$, where $(v_s \rightarrow v_e)$ is the edge traversed by Tu_w^j, and *ndist* is the network distance between v_s and the location at t.

Definition 6.3
Given a mapped location $\langle (v_s \rightarrow v_e), rd \rangle$, a probability α, and a compressed uncertain trajectory Tu^j, a **probabilistic when query when**$(Tu^j, \langle (v_s \rightarrow v_e), rd \rangle, \alpha)$ returns the set of timestamps, where rd is the relative distance of the location w.r.t. $(v_s \rightarrow v_e)$, and each timestamp t corresponds to a instance Tu_w^j of Tu^j with $Tu_w^j.p \geq \alpha$, such that Tu_w^j passed $\langle (v_s \rightarrow v_e), rd \rangle$ at t.

Definition 6.4
Given a query region RE, a timestamp t_q, and a collection of compressed uncertain trajectories $\boldsymbol{Tu}$, a **probabilistic range query range**$(\boldsymbol{Tu}, RE, t_q, \alpha)$ returns the set of uncertain trajectories $Tu^j (1 \leq j \leq M)$ in $\boldsymbol{Tu}$, such that $\sum_{Tu_w^j \in Tu^j \wedge Tu_w^j \cap RE \neq \emptyset} Tu_w^j.p \geq \alpha$ at t_q.

When querying using the StIU, we can effectively avoid unnecessary decompression by the following filtering and validating lemmas, which exploit p_{total} and $p_{\max}$ that are maintained for each reference.

Lemma 6.1
Given a query **when**$(Tu^j, \langle (v_s \rightarrow v_e), rd \rangle, \alpha)$, if $p_{\max} < \alpha$ holds for all the tuples of reference Ref_i^j in the StIU corresponding to the region where $\langle (v_s \rightarrow v_e), rd \rangle$ is located then we do not need to fully decompress Ref_i^j.

Example 6.2
Given a query **when**$(Tu^1, \langle (185191 \rightarrow 185192), 0.25 \rangle, 0.5)$ in Figure 2.4b, Ref_1^1 does not need to be fully decompressed. This is because $Ref_1^1.p_{\max}$ w.r.t. re_3 is 0.2, implying that $Nref_{1k}^1.p < 0.5\,(k = 1, 2)$.

Lemma 6.2
Given a spatial region RE, a timestamp t_q, and two edges $(v_s \rightarrow v_e)$ and $(v_{s'} \rightarrow v_{e'})$ where an uncertain trajectory instance Tu_i^j is located at timestamps t_b and $t_{b'}$ $(t_b \leq t_q \leq t_{b'})$, (i) if the subpath sp from v_s to $v_{e'}$ satisfies $sp \in RE$ then Tu_i^j overlaps RE at t_q; (ii) if the subpath sp from v_s to $v_{e'}$ satisfies $sp \cap RE = \emptyset$ then Tu_i^j does not overlap RE at t_q.

Lemma 6.3
Given a query **range**$(\boldsymbol{Tu}, RE, t_q, \alpha)$ and a set Can^j that contains the instances of $Tu^j \in \boldsymbol{Tu}$ satisfying condition (i) in Lemma 6.2, if the sum of the probabilities of all the instances in Can^j is not smaller than α then Tu^j should be in the query result.

Lemma 6.4
Given a query **range**$(\boldsymbol{Tu}, RE, t_q, \alpha)$, a region $re_{total}(RE \subseteq re_{total})$, and a set Can^j that contains all the instances of $Tu^j \in \boldsymbol{Tu}$ that overlap re_{total} during $[t_b, t_{b'}]\,(t_b \leq t_q \leq t_{b'})$, if the sum of probabilities of all the instances in Can^j is smaller than α then Tu^j does not qualify as a query result.

Example 6.3
Given a query **range**$(\mathbf{Tu}, re_3 \cup re_4$, 5:05:25, $0.5)$ in Figure 2.4 and $\eta_p = \frac{1}{2048}$, we can get the subpath from v_1 to v_4 after partially decompressing $T(Tu^1)$ and $E(Ref_1^1)$, where Ref_1^1 is located on $(v_1 \rightarrow v_2)$ at 5:03:25 and located on $(v_3 \rightarrow v_4)$ at 5:07:25. According to Lemma 6.2, we can ensure that Ref_1^1 must overlap $re_3 \cup re_4$ at 5:05:25 without decompressing $D(Ref_1^1)$. Since $Ref_1^1.p \geq 0.5$, Tu^1 can be directly returned by Lemma 6.3. Then, since the sum of the probabilities of instances in $Can^1 (= \emptyset)$ w.r.t. $re_{total} (= RE_1)$ is $0\,(< 0.5)$, Tu^1 can be safely pruned by Lemma 6.4.

7 Experimental Evaluation

7.1 Experimental Design

Datasets. We use three real-life datasets, i.e., Denmark (DK), Chengdu (CD), and Hangzhou (HZ). The DK dataset is collected from 162 vehicles over about 2 years (from Jan. 2007 to Dec. 2008) in Denmark. The CD dataset is collected from 14,864 taxis over one month (Aug. 2014) in Chengdu, China. The HZ dataset is collected from 24,515 taxis over one month (Nov. 2011) in Hangzhou, China. The DK, CD, and HZ datasets contain 0.27, 1.96, and 1.81 million NCUTs, respectively.

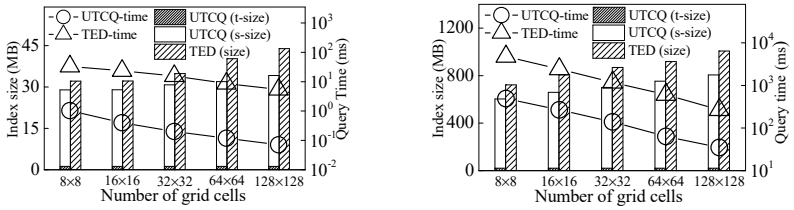
Baseline and experimental settings. Since no existing study is designed for network-constrained uncertain trajectory compression, we compare our work with TED [22], which is the state-of-the-art work for network-constrained accurate trajectory compression. All algorithms are implemented in C++ and run on a computer with an Intel Core i9-9880H CPU (2.30 GHz) and 32 GB memory.

Table 2.3: Comparison on three datasets [14]

Datasets	UTCQ						
	Compression ratio						Time(s)
	Total	T	E	D	T′	p	
Denmark	14.342	7.685	14.861	26.171	15.843	7.111	23
Chengdu	11.867	3.128	13.589	15.141	18.061	7.111	135
Hangzhou	13.787	3.193	16.092	17.815	14.592	5.818	1031

Datasets	TED						
	Compression ratio						Time(s)
	Total	T	E	D	T′	p	
Denmark	4.439	4.545	11.888	9.143	1	7.111	1823
Chengdu	4.287	1.707	11.247	9.143	1	7.111	65310
Hangzhou	4.008	1.418	9.376	9.143	1	5.818	980447

Performance metrics. We use the compression ratio (CR) and compression time to evaluate the compression performance, and we use the index size and query time to evaluate performance of querying processing.



(a) DK dataset **(b)** HZ dataset

Figure 2.5: Query performance [14]

7.2 Experimental Results

Compression performance. Table 2.3 compares UTCQ and TED, where T, E, D, T′, p, and Total refer to the *timestamps*, *outgoing edge numbers*, *relative distance*, *time flag bit-string*, *probability*, and the total compression ratio, respectively. As can be observed, UTCQ outperforms TED by more than three times in terms of compression ratio and by more than an order of magnitude in terms of compression time. This is because (i) UTCQ employs referential compression to remove more redundancy for achieving a high compression ratio and (ii) TED has to load all trajectories for the preparation of compression [22], while UTCQ compresses trajectories one by one.

Query performance. Figure 2.5 reports the performance of probabilistic range queries when varying the spatio-temporal partition granularity. It shows that the StIU index is smaller than the index used by TED, which is due to the referential compression. Next, as road networks are divided at finer granularities, the query time of both frameworks decreases. Finally, UTCQ is faster than TED, which is due to the proposed StIU index and the filtering and validation techniques.

Chapter 3

Compression of Streaming Trajectories in Road Networks

This chapter gives an overall introduction to Paper B [52]. It reuses content from that paper when this was considered most effective.

1 Problem Motivation and Statement

Most existing trajectory compression studies target offline compression [14, 17–22, 53–55]. This requires an entire trajectory is available before compression starts and is not appropriate for use by GPS-enabled devices. If trajectories are instead transmitted to a location where compression can be performed, e.g., a data center, this will result in high communication overheads or data loss. In contrast, online compression compresses GPS points once they arrive in real-time, thus enabling a broader range of applications and saving both storage and transmission costs [56–59].

Few studies target online network-constrained trajectory compression [23, 24, 56, 59] and have mainly the following two limitations. First, they obtain a compact representations by excluding important information [23, 24, 56, 59], e.g., the exact locations of trajectories. This reduces the accuracy of the compressed trajectories and thus decreases their usability. Second, previous studies perform online compression on top of prediction models trained offline using historical data [23, 24]. However, movement patterns vary across time [25, 26]. This necessitates frequent re-training and delivery of re-trained models, which increases the communication overheads.

To address the above two limitations, we propose a novel framework for online TRAjectory ComprEssion (TRACE). We first present a speed-based trajectory representation and a multiple-reference based referential representation for improving compression. Next, we develop an effective online reference selection technique based on so-called k-mer matching [60–62]. A reference deletion algorithm is proposed to keep the memory consumption low, while a reference rewriting algorithm is proposed to adapt to varying movement patterns. Further, we provide a data transmission strategy that enables low-overhead transmission of trajectories in real-time. Finally, we develop an index structure and filtering techniques that facilitate real-time range querying of compressed trajectories.

In summary, our main contributions are as follows:

- We propose a new real-time streaming vehicle trajectory compression, transmission, and querying framework.
- We develop a speed-based representation and a multiple-reference based referential representation. We present an online reference selection technique together with reference deletion and rewriting functions.
- We present a data transmission scheme that reduces transmission overhead. We also propose an index structure and filtering techniques to accelerate real-time query processing.
- Extensive experiments offer insight in the TRACE and also show that it is capable of outperforming three baselines in terms of both compression ratio and transmission cost.

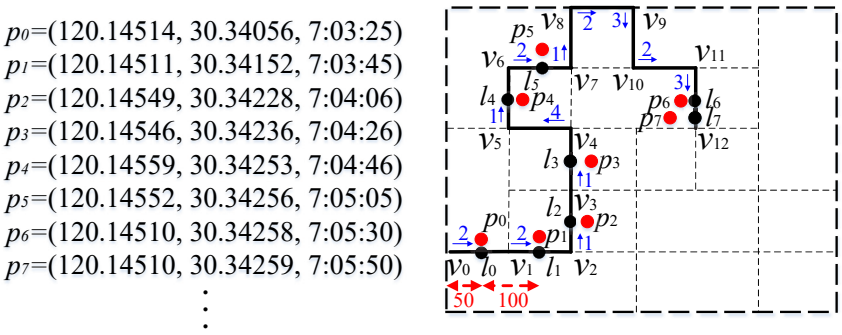
2 Preliminaries

In this chapter, we formally define streaming trajectories in road networks. The following definitions and examples are reproduced from [14].

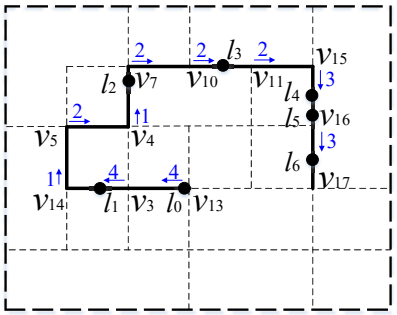
A **path** sp is a sequence of connected edges $(v_i \rightarrow v_j)$ that starts from v_s and ends at v_e, i.e., $sp = \langle (v_s \rightarrow v_0), \cdots, (v_{n-1} \rightarrow v_e) \rangle$.

Definition 2.1
A **streaming network-constrained trajectory** Tr^n is modeled as an infinite, time-ordered sequence of mapped GPS points L^n with an infinite path $sp(Tr^n)$ traversed by Tr^n.

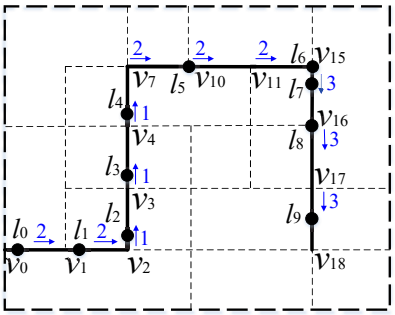
Definition 2.2
The **accumulative distance** of a streaming trajectory Tr^n at its i^{th} timestamp $t(Tr^n)[i]$, denoted as $ad(Tr^n)[i]$, is the network distance between v_s and l_i along the path $(v_s \rightarrow v_e), \cdots, (v_{s^*} \rightarrow v_{e^*})$, where l_i is located on $(v_{s^*} \rightarrow v_{e^*})$ and $(v_s \rightarrow v_e)$ is the first edge traversed by Tr^n. The accumulative distance sequence $ad(Tr^n)$ of a streaming trajectory Tr^n contains the trajectory's accumulative distance at each timestamp.



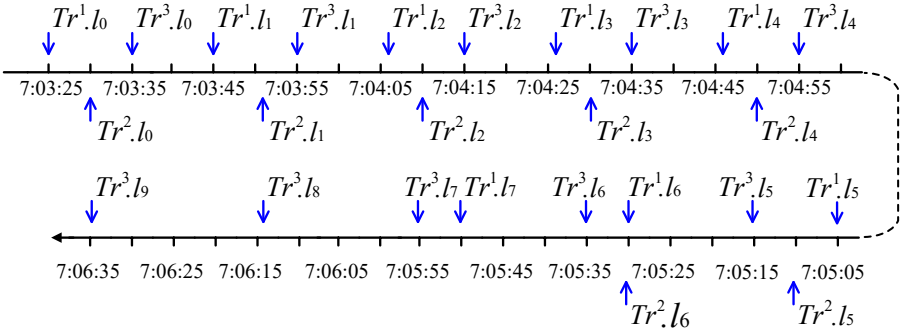
(a) Tr^1



(b) Tr^2



(c) Tr^3



(d) Time line of Tr^1, Tr^2 and Tr^3

Figure 3.1: A streaming network-constrained trajectory set $\mathbf{Tr} = \{Tr^1, Tr^2, Tr^3\}$ [52]

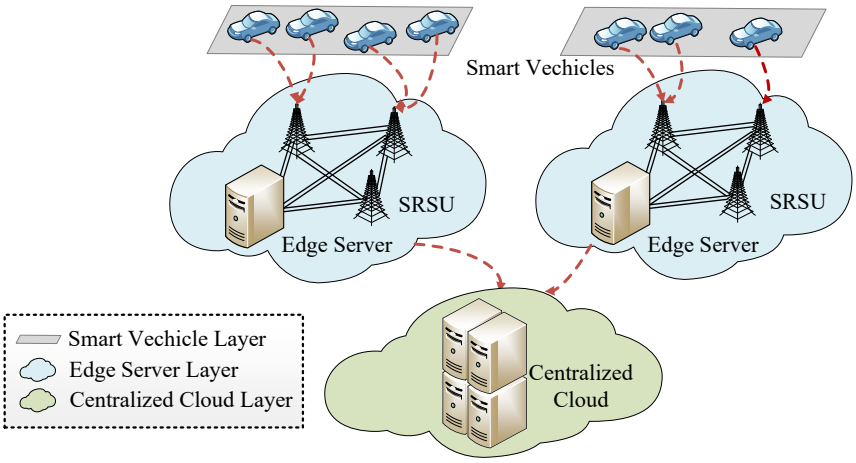


Figure 3.2: Vehicular edge computing architecture [52]

Example 2.1
Figure 3.1 shows a network-constrained trajectory set $\mathbf{Tr} = \{Tr^1, Tr^2, Tr^3\}$ that contains three streaming network-constrained trajectories. $sp(Tr^1) = \langle (v_0 \rightarrow v_1), \cdots, (v_{11} \rightarrow v_{12}), \cdots \rangle$ and $L^1 = \{l_0, l_1, \cdots, l_7, \cdots\}$. Given that the network distance between v_0 and l_1 is 150, $ad(Tr^1)[1] = 150$.

Since we propose a multiple-reference based referential representation, a represented non-reference *Nref*, defined in Definition 4.1 in Chapter 2 is denoted as $Com_\phi(Nref)$ in this chapter.

3 Framework

TRACE enables online compression and subsequent querying of streaming network-constrained trajectories. To enable real-time compression on diverse devices with variable computing capabilities, we employ vehicular edge computing (VEC) [56], as shown in Figure 3.2. The edge server layer is near the smart vehicle layer, but is generally far away from the centralized cloud layer. TRACE is deployed at both the edge server layer and the centralized cloud layer. Specifically, TRACE takes network-constrained streaming trajectories as input, which are delivered from the smart vehicle layer. Next, TRACE performs trajectory representation and compression at the edge server layer and transfers the compressed trajectories to the centralized cloud layer for query processing. This way, complex computations are accomplished at the edge server layer, while only compressed trajectories are subject to long-distance transmission. Hence, both wireless communication energy consumption and network load [63, 64] are reduced.

Table 3.1: Speed-based representation of **Tr** in Figure 3.1 [52]

n	1	2	3
$SV(Tr^n)$	44183	27444	44183
$E(Tr^n)$	$\langle 2,2,1,1,4,1,2,1,2,3,2,3\rangle$	$\langle 4,4,1,2,1,2,2,2,3,3\rangle$	$\langle 2,2,1,1,1,2,2,2,3,3,3\rangle$
$RD(Tr^n)$	0.5	0.05	0.25
$GD(Tr^n)$	$\langle 0.67, 0.5, 0.5, 0.67$ $0.33, 0.83, 0.05\ \rangle$	$\langle 0.97, 0.69, 0.35, \rangle$ $0.53, 0.11, 0.75\rangle$	$\langle 0.8, 0.5, 0.5, 0.5, 0.64\rangle$ $0.53, 0.11, 0.75, 0.67\rangle$
$V(Tr^n)$	$\langle 5, 4.76, 5, 10, \rangle$ $5.26, 20, 1.25\rangle$	$\langle 6.9, 17.11, 8.75, \rangle$ $\langle 10, 1.25, 3.75\rangle$	$\langle 5, 5, 5, 5, 8.75, \rangle$ $\langle 10, 1.25, 3.95, 7.5\rangle$
$T(Tr^n)$	$\langle$7:03:25, 0, 1, $0, 0,$-1, $5, 0\rangle$	$\langle$7:03:30, 1, -1 $0,0,0,0\rangle$	$\langle$7:03:35, 0, 0, 0, $0,0,0,0,-1,0\rangle$

4 Representation

This chapter covers the key techniques of trajectory representation. The following definitions and examples are reproduced from [52].

4.1 Speed-based Representation

Co-movement patterns of trajectories motivate us to apply referential representation to speeds. We thus propose a speed-based representation, where Tr is represented as a tuple $(SV(Tr), E(Tr), RD(Tr), GD(Tr), V(Tr), T(Tr))$. $GD(Tr)[i]$ is the growth rate of the accumulative distance at $T(Tr)[i]$, calculated as $\frac{ad(Tr)[i]-ad(Tr)[i-1]}{ad(Tr)[i]-ad(Tr)[i-2]}$. $V(Tr)[i]$ is the speed, computed as $\frac{ad(Tr)[i]-ad(Tr)[i-1]}{t(Tr)[i]-t(Tr)[i-1]}$ $(i > 0)$, where $t(Tr)[i]$ is the i^{th} timestamp of Tr. An example of the speed-based representation is given in Table 3.1.

4.2 Multiple-reference based Referential Representation

Next, we apply referential compression [14] to the speed-based representation. Specifically, we modify the (S, L, M) format to the (ref_{id}, S, L, M) format to enable the use of multiple references, where ref_{id} is the ID of a reference. The referential representation of $E(Tr)$ essentially follows the UTCQ representation [52]. However, unlike the out-going edge numbers, two speeds are unlikely to be exactly the same. We thus consider $V(Tr^n)[i] \approx V(Tr^{n'})[i']$ if $\dot{V}(Tr^n)[i] = \dot{V}(Tr^{n'})[i']$, where $\dot{V}(Tr^n)[i]$ is the integer closest to $\frac{V(Tr^n)[i]}{0.5^\eta}$, and η is the speed error bound. If $V(Tr)$ is a non-reference, M is recorded as $GD(Tr)[i]$ if we encounter a mis-matched value $V(Tr)[i]$; otherwise, we just keep $GD(Tr)$ and discard $V(Tr)$. The reason is that $V(Tr)[i]$ $(\in [0, \mu])$ can only achieve the same compression as $GD(Tr)[i]$ $(\in [0, 1))$ at the cost of compression accuracy, where μ is a speed constraint of the road network. Among the elements of the speed-based representation of Tr, only $E(Tr)[i]$ and $V(Tr)[i]$ are referentially represented.

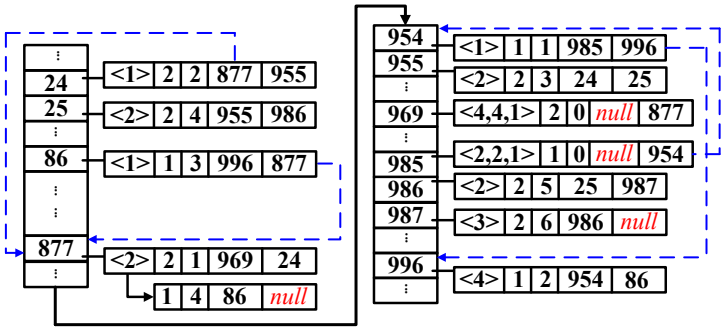


Figure 3.3: A hash table $\boldsymbol{H}$ constructed according to E(Tr) in Table 3.1 at 7:05:06, where $\boldsymbol{k}$**=3** [52]

Example 4.1
Let $\eta = 0$. In Table 3.1, the referential representation of $E(Tr^3)$ w.r.t. $E(Tr^1)$ and $E(Tr^2)$ is $Com_E(Tr^3) = \langle(1,0,4,1),(2,5,5,3)\rangle$, while that of $V(Tr^3)$ w.r.t. $V(Tr^1)$ and $V(Tr^2)$ is $Com_V(Tr^3) = \langle(1,0,3,0.5),(2,2,4,0.67)\rangle$.

4.3 Reference Selection and Deletion

As the online reference selection and deletion for $E(Tr)$ and that for $V(Tr)$ are very similar, we only discuss $E(Tr)$.

Reference selection. Following an existing study [61], we start by decomposing a subsequence into k-mers. A ***k*-mer** ϕ_i^n is a subsequence of fixed length k of $\phi(Tr^n)$, i.e., $\phi_i^n = \{\phi(Tr^n)[i], \phi(Tr^n)[i+1], ..., \phi(Tr^n[i+k-1])\}$.

Definition 4.1
The subsequence of $E(Tr)$ of a streaming trajectory Tr arriving at $t(Tr)[j]$ $(j > 0)$, is denoted as $E(Tr_j)$ $(j > 0)$. It starts from the edge traversed by Tr immediately after leaving the last edge of $E(Tr_{j-1})$ and ends at the edge where Tr is located at $t(Tr)[j]$.

Example 4.2
In Figure 3.1a, we have $E(Tr_4^1) = \langle 4,1\rangle$ because $(v_4 \to v_5)$ is traversed by Tr^1 after leaving $(v_3 \to v_4)$ and l_4 is located on $(v_5 \to v_6)$ at $T(Tr^1)[4]$.

Next, we construct a hash table H for $E(Tr^n)$. Each k-mer in H is stored as a tuple $(E_i^n, n, off_i, pt_i, pd_i)$, where i) E_i^n is the i^{th} k-mer of $E(Tr^n)$; ii) n is the ID of Tr^n; iii) off_i is the offset of $E_i^n[0]$ in $E(Tr^n)$; and iv) pt_i and pd_i are the indexes of the entries associated with E_{i-1}^n and E_{i+1}^n in H, respectively. The hash table built according to the streaming trajectories in Table 3.1 is shown in Figure 3.3. As can be seen, we only store $E_i^n[k-1]$ $(i > 0)$ rather than E_i^n to save space. This is because the whole k-mer can be retrieved by pt_i.

Example 4.3
In Figure 3.3, the entry associated with E_1^1 $(= (2,1,1))$ in H is stored as ($\langle 1\rangle$, 1, 1, 985, 996), where 985 and 996 are the indexes of E_0^1 and E_2^1 of Tr^1 in H.

A k-mer is only formed when $|E(Tr_m^n) \cup E(Tr_{m+1}^n) \cup \cdots \cup E(Tr_{m+j}^n)|$ is no less than k. Once a k-mer is formed, we calculate its hash key, key, and hash it to H. There are two cases: $H[key] = \emptyset$ and $H[key] \neq \emptyset$. The former implies that no identical subsequence is already stored in H, where the corresponding sequence $E(Tr_m^n)$ is assigned as a reference. The later implies that the subsequence can be represented by an existing reference; thus, $E(Tr_m^n)$ is assigned as a non-reference.

Example 4.4
Continuing the example in Figure 3.1 and Table 3.1, since E_0^3 matches E_0^1, we initialize a factor $(1, 0, 3, \emptyset)$ for it. For the arriving $E(Tr_3^3) = \langle 1 \rangle$, we retrieve $E(Tr^1)[3]$ according to the index pd_0 associated with E_0^1, i.e, 954, and compare it with $E(Tr_3^3)$ $(= E_1^3[2])$. After that, we update the factor to $(1, 0, 4, \emptyset)$ due to $E(Tr^1)[3] = E(Tr_3^3)$. As $E(Tr^1)[4] \neq E_2^3[2]$, we generate a factor, i.e., $(1, 0, 4, 1)$, for $Com_E(Tr^3)$. Then, we wait until $E(Tr_6^3)$ arrives due to $|E(Tr_5^3) \cup E(Tr_6^3)| \geq 3$ and repeat the process. Finally, we get $Com_E(Tr^3) = \langle (1, 0, 4, 1), (2, 5, 5, 3) \rangle$.

Reference selection. The number of references stored in the hash table increase over time, which motivates us to eliminate outdated references. We consider a reference to be outdated if it has not been visited for a long time. Specifically, a trajectory is visited at timestamp t if (i) it is used for referential representation and/or (ii) its corresponding data still arrives. We define G_o as the set of the references at timestamp t_o and denote each reference in G_o, $G_o[i]$, as a tuple $(ref_{id}, G_o[i].tl)$, where ref_{id} is the ID of the reference $G_o[i]$, and $G_o[i].tl$ is the timestamp when $G_o[i]$ was visited most recently.

Definition 4.2
A reference $G_o[i]$ is **outdated** at t_o if its **freshness** at t_o, denoted as $G_o[i].f$, satisfies $G_o[i].f < C \cdot \frac{F_o}{|G_o|}$, where $F_o = \sum_{i'=0}^{|G_o|-1} G_o[i'].f$ and C $(0 < C \leq 1)$ is the deletion coefficient.

The freshness $G_o[i].f$ of a reference $G_o[i]$ is calculated as follows:

$$G_o[i].f = \lambda^{t_o - G_o[i].tl} \quad (t_o \geq G_o[i].tl), \tag{3.1}$$

where $\lambda \in (0, 1)$ is a decay factor [65].

Example 4.5
In Figure 3.1, given the current timestamp t_o =7:03:26 and $\lambda = 0.998$, we get $E(Tr^1).f = 0.998$ at t_o.

According to Definition 4.2 and Formula 3.1, we can update the freshness of each reference to determine whether it is outdated. However, this naive strategy is time-consuming, especially when the number of references stored in the hash table is large. Thus, we propose to get the sum of all references at timestamp t_o, i.e., F_o, by the following formula.

$$F_o = (F_{o'} - \sum_{G_{o'}[i] \in Rv_o} G_{o'}[i].f) \cdot \lambda^{t_o - t_{o'}} + |Rv_o|, \tag{3.2}$$

where Rv_o ($Rv_o \subseteq G_o$) is a set of references visited at t_o. With Formula 3.2, we are enable to update F_o by only computing the freshness of $G_{o'}[i]$, such that $G_{o'}[i] \in Rv_o$.

4.4 Reference Rewriting

We rewrite references in real-time to further improve the compression ratio. A motivating example is shown as follows.

Example 4.6
If the subsequence of $E(Tr^3)$, $\langle 44183, 2, 2, 1, 1, 1, 2, 2\rangle$, is frequent, $\langle 2, 2, 1, 1, 1, 2, 2\rangle$ should be stored as k-mers. However, $E(Tr^3)$ is assigned as a non-reference just because $E(Tr^1)$ arrives earlier than it, as shown in Example 4.4. In this case, for another new arriving subsequent Tr^n that also traverses $\langle 44183, 2, 2, 1, 1, 1, 2, 2\rangle$, $Com_E(Tr^n)$ will contain at least two factors.

Example 4.6 illustrates that it is attractive to update infrequent references in real-time, which is achieved by identifying rewriting candidates.

Definition 4.3
A **rewriting candidate** is a reference $E(Ref)$ that represents a non-reference $E(Nref)$ as $Com_E(Nref) = \langle \cdots, (ref_{id}, S, L, M), (ref_{id}, S', L', M'), \cdots \rangle$, where $S + L + 1 = S'$ and ref_{id} is the ID of the reference Ref.

In Definition 4.3, we identify $E(Ref)$ as a rewriting candidate because two factors (ref_{id}, S, L, M) and (ref_{id}, S', L', M') can be merged if $E(Ref)[S + L]$ is replaced with M, which is called a rewriting operation. Here we only consider rewriting $E(Ref)$ because the patterns of speed are very likely to vary accross different time periods [25, 26]. Given a factor (ref_{id}, S, L, M), it intersects $E(Ref)[i]$, if ref_{id} is the ID of Ref and $S \leq i < S + L$.

There are two principles for conducting a rewriting operation. First, M should occur frequently, to make sure that the rewritten reference is a frequent subsequence. Second, the factors intersecting $E(Ref)[i]$ should occur infrequently; otherwise, many factors may be separated when applying the rewritten result to the referential representation. These principles require us to record each factor represented by $E(Ref)$, if it is identified as a rewriting candidate. This leads to a high space-time consumption.

Inspired by a regular square grid graph [66], we construct a $b \times b$ factor matrix **FA** for each $E(Ref)$, where $b = \lceil \frac{|E(Ref)|_r}{k} \rceil$ and $|E(Ref)|_r$ is the length of the subsequence of $E(Ref)$ used as a reference. We denote an element $FA[x][y]$ in **FA** as FA^{xy}, which corresponds to the subsequence $\langle E(Ref)[(x -$

$1) \cdot k], \cdots, E(Ref)[y \cdot k - 1]\rangle$. A factor (ref_{id}, S, L, M) contributes to FA^{xy}, where $x = \lfloor \frac{S}{k} \rfloor + 1$ and $y = \lceil \frac{S+L}{k} \rceil$.

Example 4.7
Consider $E(Tr^4) = \langle 2,2,1,1,4,1,3,1,2,3,3\rangle$, $E(Tr^5) = \langle 1,1,4,1,3,1,2,3,3\rangle$, and $E(Tr^6) = \langle 2,2,1,1,4,1,2,1,2,2,2,2,1,3\rangle$. Following Example 4.4 and assuming that $\mathbf{Tr} = \{Tr^1, Tr^2, Tr^3, Tr^4, Tr^5, Tr^6\}$, the factor matrix **FA** of $E(Tr^1)$ is:

$$\mathbf{FA} = \begin{bmatrix} [0,2] & [0,5] & [0,8] & [0,11] \\ & [3,5] & [3,8] & [3,11] \\ & & [6,8] & [6,11] \\ & & & [9,11] \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 3 & 1 & 0 \\ & 0 & 0 & 0 \\ & & 0 & 2 \\ & & & 0 \end{bmatrix}$$

Here, the left part of **FA** intuitively gives the subsequences of $E(Tr^1)$ corresponding to FA^{xy}, and the right part shows the value of FA^{xy}. For instance, $FA^{12} = 3$ is contributed by three factors, i.e., $(1,0,4,1)$, $(1,0,6,3)$, and $(1,2,4,3)$.

Lemma 4.1
Given a factor (ref_{id}, S, L, M) intersecting $E(Ref)[i]$ and a $b \times b$ factor matrix **FA** of $E(Ref)$, (ref_{id}, S, L, M) can only contribute to FA^{xy}, where $x \leq \frac{i}{k} + 1 \wedge y > \frac{i}{k}$.

Lemma 4.2
Given a $b \times b$ factor matrix **FA** of $E(Ref)$, the maximum number of factors that intersect $E(Ref)[i]$ is $\sum_{x=1}^{\lfloor \frac{i}{k} \rfloor + 1} \sum_{y=\lfloor \frac{i}{k} \rfloor + 1}^{b} FA^{xy}$.

Example 4.8
Continuing Example 4.4, the maximum number of the factors intersecting $E(Tr)[6]$ is $\sum_{x=1}^{3} \sum_{y=3}^{4} FA^{xy} = 3$, where only 6 out of 10 elements in FA need to be visited.

Example 4.8 implies that Lemmas 4.1 and 4.2 enable rewriting by scanning part of the factor matrix, which enhances the efficiency of reference rewriting. Based on the above conclusions, we formulate the conditions for implementing a rewriting operation, i.e, replacing $E(Ref)[i]$ with M: i) the frequency of occurrence of M, $f(M)$, is no less than a threshold α, and ii) $\sum_{x=1}^{\lfloor \frac{i}{k} \rfloor + 1} \sum_{y=\lfloor \frac{i}{k} \rfloor + 1}^{b} FA^{xy} < f(M)$.

5 Compression

This chapter covers the binary encoding and data transmission schemes for compressed trajectories. The binary code of *seq* is denoted as *sêq*. The following formula and example are reproduced from [52].

5.1 Binary Encoding

We propose to encode the floating number in $RD(Ref)$ and $GD(Ref)$ by the following formula.

$$\hat{fv} = \arg\min_{\hat{fv}_m} \left| \sum_{i=1}^{|\hat{fv}_m|} \hat{fv}_m[i-1] \cdot \frac{1}{2^i} - fv \right|, \tag{3.3}$$

where $|\hat{fv}_m| = \gamma$ and γ is a threshold that controls the compression accuracy. We set the lengths of $\hat{S}$ and $\hat{L}$ to be the same and adopt variable-length encoding [14] for referential compression. Moreover, we record the lengths of $\hat{S}$ and $\hat{L}$ for decompression, as the whole length of a sequence $\phi(Tr)$ ($\phi = E, V$) is unknown in streaming settings.

5.2 Data Transmission

We propose a data transmission strategy that targets low transmission cost and enables decoding at the centralized cloud. The strategy contains three states: ① transferring the initialized factor $(\hat{ref}_{id}, S, L', \emptyset)$, where $L' \geq k$ and $|\hat{S}| = |\hat{L'}|$; ② transferring the updated L'; and ③ transferring the mis-matched element M when L' is updated to L. Setting $\gamma \geq 2$ and $2 \cdot |\hat{S}| \neq |\hat{M}|$, the binary codes delivered at each state can be distinguished just by their lengths; thus, no extra information needs to be transmitted. Example 5.1 illustrates this.

Example 5.1
Continuing Example 4.7 and letting both $\hat{ref}_{id}$ (≥ 1) and $\hat{M}$ take 3 bits, the first factor of $Com_E(Tr^6)$ is initialized as $(1, 0, 3, \emptyset)$ and thus is encoded as $(000, 00, 10)$. Then $\hat{bc} = 0000010$ is sent to the centralized cloud, which triggers the state transition from ③ to ①. Next, we continue to transfer $\hat{bc} = 0001$ before the mis-matched value $M = 2$ is found, where the first three bits is to record $\hat{ref}_{id}$. During this process, the state is first transferred to ② and then remains unchanged. Meanwhile, L' continues to be incremented by 1. Once the centralized cloud receives $M = 2$, i.e., $\hat{bc} = 000001$, a factor is generated and stored in the form $(\hat{M}, \hat{len}, \hat{ref}_{id}, \hat{S}, \hat{L})$, i.e., as $(001, 0, 000, 0000, 1000)$, in the centralized cloud.

6 Query Processing

This chapter covers an index and accompanying filtering techniques for facilitating real-time range queries. The definitions and examples are reproduced from [52].

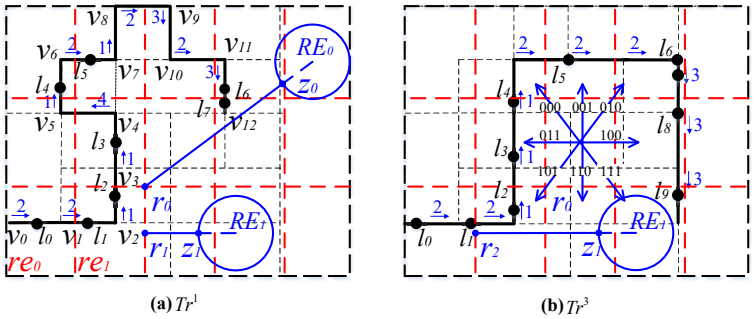


Figure 3.4: The partition of the road network G in Figure 3.1 [52]

Definition 6.1
Given a query region RE and a set of compressed streaming trajectories $\hat{\boldsymbol{Tr}}$, a **range query range**$(\hat{\boldsymbol{Tr}}, RE)$ returns the set of streaming trajectories $Tr^n (1 \leq n \leq N)$ in $\hat{\boldsymbol{Tr}}$, such that $Tr^n \cap RE \neq \emptyset$ at the current timestamp.

We partition the road network G using grid cells, each of which is a region re and links to the streaming trajectories that are currently located in it. Figure 3.4 partitions the road network G in Figure 3.1.

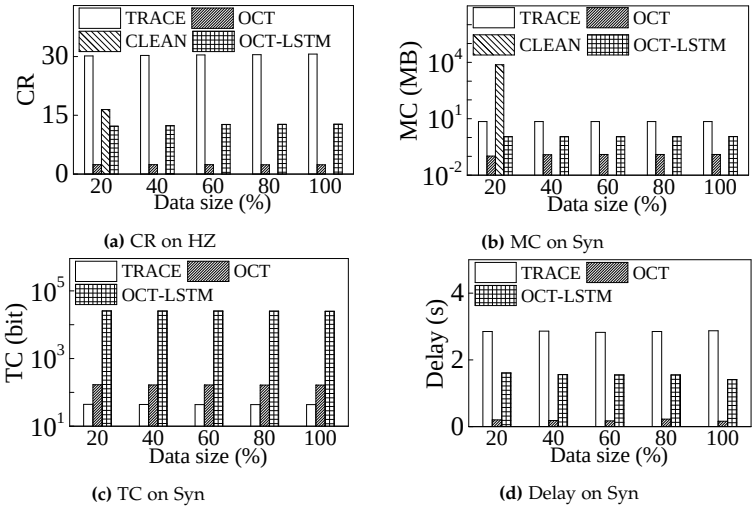
Definition 6.2
The **minimum distance** $mind(re, RE)$ between a grid cell re and a query region RE is the distance between a location r and a location z, denoted as $|rz|$, where $r \in re \wedge z \in RE \wedge \forall r' \neq r\ (r' \in re \wedge |r'z| \geq |rz|) \wedge \forall z' \neq z\ (z' \in RE \wedge |rz'| \geq |rz|)$.

Lemma 6.1
Given a range query **range**$(\hat{\boldsymbol{Tr}}, RE)$, the current timestamp tc, the reachable distance dis of Tr^n w.r.t. tc, and Tr^n located in grid cell re at $Tr^n.tp$, if the minimum distance $mind(re, RE) > dis$, Tr^n cannot be in the result.

Lemma 6.1 enables pruning Tr^n without computing its location at the current timestamp, which is illustrated in Example 6.1.

Example 6.1
Given $|r_2 z_1| = 233$ in Figure 3.4b, we do not need to decompress $\hat{Com}_E(Tr^3)$ and $\hat{Com}_V(Tr^3)$ if a range query ***range***$(\hat{\mathbf{Tr}}, RE_1)$ arrives at 7:04:06. This is because Tr^3 overlaps re_0 at 7:03:55 and the reachable distance of Tr^3 w.r.t. 7:04:06 is 231<233.

7 Experimental Evaluation



(a) CR on HZ **(b)** MC on Syn **(c)** TC on Syn **(d)** Delay on Syn

Figure 3.5: Comparison [52]

7.1 Experimental Design

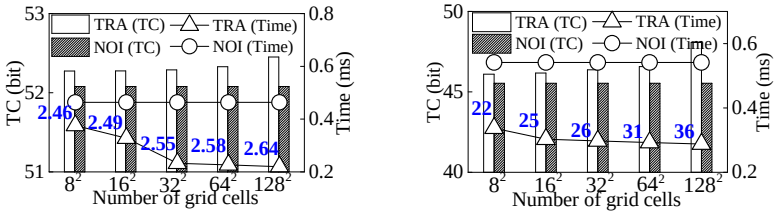
Datasets. The experiments use two real datasets, Denmark (DK) and Hangzhou (HZ), and a synthetic dataset, Synthetic (Syn). DK, HZ, and Syn contain 0.42, 1.92, and 50 million trajectories, respectively.

Baselines and experimental settings. We use three baselines: CLEAN [55], OCT-LSTM [24], and OCT [24]. CLEAN targets trajectory compression in offline settings and exploits movement patterns of trajectories to improve compression. OCT-LSTM and OCT are online methods. OCT-LSTM trains an LSTM model that mines repetitive patterns of time-distance sequences using historical data and performs compression by discarding data that cannot be predicted accurately. OCT [24] employs a linear prediction model and does not involves offline training. All algorithms are implemented in C++ and run on a computer with an Intel Core i9-9880H CPU (2.30 GHz) and 32 GB memory.

Performance metrics. We use the compression ratio (CR), time delay (Delay), maximum memory cost (MC), and transmission cost (TC) to evaluate compression performance, and we use the query time (Time) and transmission cost (TC) to evaluate query performance.

7.2 Experimental Results

Compression performance. Figure 3.5 reports experimental results when varying the size of a dataset from 20% to 100%. Since 80%, 10%, and 10% data of each dataset are used for training, validation, and testing, respectively, for OCT-LSTM, all methods are performed on a 10% dataset. Since CLEAN is an offline method and takes 192.1 hours to compress 20% Syn, we only



(a) TC and query time on DK

(b) TC and query time on HZ

Figure 3.6: Query performance [52]

report its compression ratio and maximum memory cost on HZ and 20% Syn. First, TRACE outperforms all the baselines in terms of compression ratio and transmission cost, due to its two-stage representation and its encoding and transmission schemes. Next, the maximum memory cost and time delay of TRACE are higher than those of OCT-LSTM and OCT. This is because TRACE trades these for higher compression ratios and lower transmission cost, by maintaining a reference table and employing k-mers matching.

Query performance. Figure 3.6 reports the transmission cost and query time when varying the number of grid cells. "TRA" denotes our TRACE framework while "NOI" denotes the case of no indexing. Intuitively, a fine-grained partition of road networks leads to higher query efficiency and larger transmission cost. The blue numbers along the TRACE query time line denote the index creation time (μs), which is the average time used on creating/updating indexes for all the arriving locations at each timestamp. Clearly, this time is negligible compared with the query time.

Chapter 4

Evolutionary Clustering of Streaming Trajectories

This chapter gives an overall introduction to Paper C [67]. It reuses content from that paper when this was considered most effective.

1 Problem Motivation and Statement

The clustering of streaming trajectories facilitates finding representative paths or movement trends that are shared by objects in real time [30–38]. Existing real-time clustering studies perform clustering using only the most recent data and target high clustering efficiency at the expense of clustering quality [68]. However, clusters should be robust to short-term fluctuations in the underlying trajectory data, which may be achived by means of smoothing [39]. We use an example to illustrate this.

Example 1.1
Figure 4.1 shows the trajectories of 12 moving objects at three timestamps, $k = 1, 2, 3$. Traditional clustering algorithms return clusters $c_1 = \{o_1, o_2, o_3, o_4, o_5, o_6\}$ and $c_2 = \{o_7, o_8, o_9, o_{10}, o_{11}, o_{12}\}$ at the first timestamp, clusters $c_1 = \{o_1, o_2, o_3, o_4, o_5\}$, $c_2 = \{o_7, o_8, o_9, o_{11}\}$, and $c_3 = \{o_6, o_{10}, o_{12}\}$ at the second timestamp, and the same two clusters at the third timestamp as at the first timestamp.

The fluctuation of the clustering results may be caused by errors related to o_6 and o_{10} at the second timestamp. Such errors may occur because GPS devices are sensitive to external factors [4]. It is clear that returning a consistent and robust clustering over all three timestamps is preferable. A intuitive strategy to achieve this is to correct the GPS records of the two objects before performing the clustering. However, studies on two real-life datasets suggest

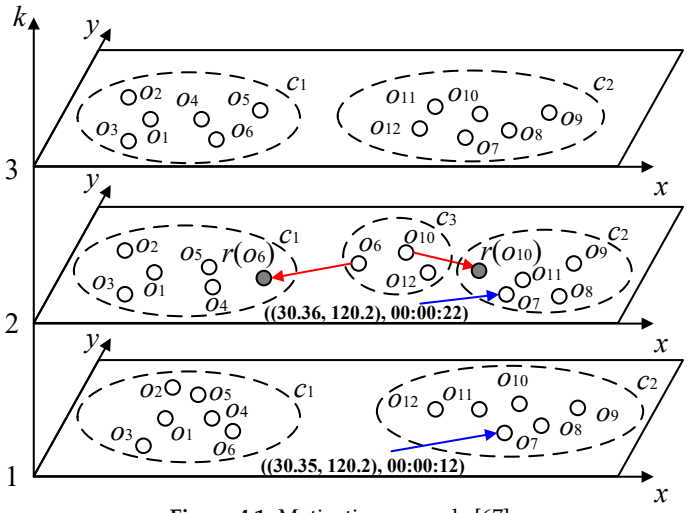


Figure 4.1: Motivating example [67]

that among the trajectories that incur the mutations of clusterings, 88.9% and 75.9% of the trajectories comply with speed constraints, while 97.8% and 96.1% of the trajectories are not outliers [42]. Further, it is infeasible to correct previous clusterings retroactively in real-time applications. Thus, existing trajectory cleaning techniques [43–45] are unsuitable for smoothing.

Evolutionary clustering [39, 68–76] has been proposed for smoothing clustering results in online scenarios. It evaluates clustering quality in terms of so-called snapshot and temporal costs, thus taking *temporal smoothness* into account. However, existing studies of evolutionary clustering are not suitable for smoothing clustering of trajectories. First, they generally target dynamic networks, which differ substantially from two-dimensional trajectory data. Second, most of them smooth clustering results iteratively until convergence, which is infeasible in settings with large-scale streaming trajectories.

We propose an efficient and effective method for evolutionary clustering of streaming trajectories (**ECO**). First, we develop a structure called *minimal group* to summarize trajectories close to each other as the basis for smoothing. Second, we present new notions of snapshot cost and historical cost and integrate these into an optimization problem, akin to what is done in existing studies [68, 72, 74, 76–78]. Next, we present a linear solution to the proposed optimization problem based on a set of mathematical proofs. Finally, we introduce a grid index structure together with an algorithm for evolutionary clustering. The paper's main contributions are summarized as follows:

- We formalize the ECO problem. To the best of our knowledge, this is the first proposal for streaming trajectory clustering that takes temporal smoothness into consideration.
- We formulate ECO as an optimization problem, based on the new no-

tions of snapshot cost and historical cost. We prove that the optimization problem can be solved approximately in linear time.
- We propose a *minimal group* structure to facilitate temporal smoothing and a *seed point shifting* strategy to improve clustering quality.
- We conduct extensive experiments on two real-life datasets that offer insight into the properties of ECO and show that ECO is capable of outperforming state-of-the-art proposals in terms of both clustering quality and efficiency.

2 Preliminaries

We introduce preliminary definitions, DBSCAN, and evolutionary clustering. The following definitions and examples are reproduced from [67].

2.1 Data Model

Definition 2.1
A **GPS record** is a pair (l, t), where t is a **timestamp** and $l = (x, y)$ is a **location**, with x being a longitude and y being a latitude.

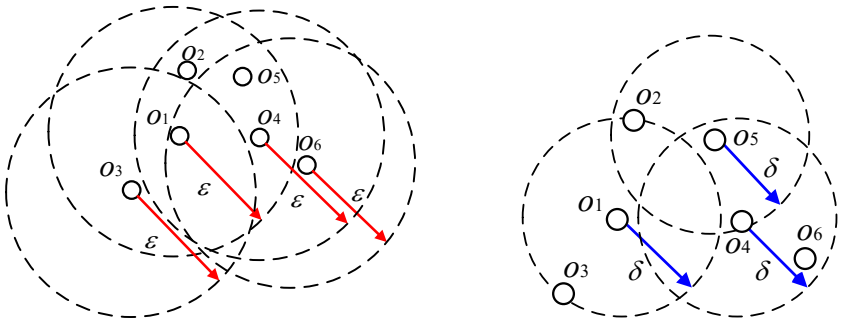
Definition 2.2
A **streaming GPS trajectory** o is an unbounded ordered sequence of timestamped, mapped locations, $\langle (o.l_1, o.t_1), (o.l_2, o.t_2) \cdots \rangle$.

To facilitate the subsequent processing, we discretize time into short intervals that are indexed by integers, following an existing study [36]. Given a time interval, the timestamp of each GPS record is mapped to the index of the interval that the timestamp belongs to. We call each index a **time step** dt.

Example 2.1
Assume that we partition the time domain into intervals of duration $\Delta t = 10s$. The two time series $\langle$00:00:01, 00:00:12, 00:00:20, 00:00:31, 00:00:44$\rangle$ and $\langle$00:00:00, 00:00:13, 00:00:21, 00:00:31, 00:00:40$\rangle$ are both mapped to $\langle 0, 1, 2, 3, 4 \rangle$.

A trajectory is **active** at time step $dt = [t_1, t_2]$ if it contains a GPS record (l, t) such that $t \in [t_1, t_2]$. A **snapshot** $\mathcal{O}_k$ is the set of trajectories that are active at time step dt_k. We denote the GPS record of o at dt_k as $(o.l_k, o.t_k)$. In the case of the current time step dt_k, $o.l_k$, $o.t_k$, $o.l_{k-1}$ and, $o.t_{k-1}$ are simplified as $o.l$, $o.t$, $o.\tilde{l}$ and $o.\tilde{t}$, respectively.

Definition 2.3
A **θ-neighbor set** of a streaming trajectory $o\,(\in \mathcal{O}_k)$ at the time step dt_k is $\mathcal{N}_\theta(o) = \{o' | o' \in \mathcal{O}_k \wedge d(o.l, o'.l) \leq \theta\}$, where $d(\cdot)$ is Euclidean distance and θ is a distance threshold. $|\mathcal{N}_\theta(o)|$ is called the **local density** of o w.r.t. θ at dt_k.



(a) Core points o_i $(1 \leq i \leq 6 \wedge i \neq 3)$ $(minPts = 3)$ **(b)** Seed points o_1 and o_4 $(\rho = 3)$

Figure 4.2: o_i $(1 \leq i \leq 6)$ at dt_1 in Figure 4.1 [67]

Example 2.2
Figure 4.1 shows three snapshots $\mathcal{O}_1$, $\mathcal{O}_2$, and $\mathcal{O}_3$. We have $o_7.\tilde{l} = o_7.l_1 = (30.35, 120.2)$, $o_7.\tilde{t} = o_7.t_1 =$ 00:00:12, $o_7.l = o_7.l_2 = (30.36, 120.2)$, and $o_7.t = o_7.t_2 =$ 00:00:22 at dt_2. In Figure 4.2, $\mathcal{N}_\delta(o_1) = \{o_1, o_2, o_3\}$.

2.2 DBSCAN

We adopt a well-known density-based clustering approach, DBSCAN [79], for clustering. The following concepts are from [79].

Definition 2.4
A trajectory $o \in \mathcal{O}_k$ is a **core point** w.r.t. ε and $minPts$, if $\mathcal{N}_\varepsilon(o) \geq minPts$.

Definition 2.5
A trajectory $o \in \mathcal{O}_k$ is **density reachable** from another trajectory $o' \in \mathcal{O}_k$, if a sequence of trajectories $o_1, o_2, \cdots o_n$ $(n \geq 2)$ exists such that (i) $o_1 = o'$ and $o_n = o$; (ii) o_w $(1 \leq w < n)$ are core points; and (iii) $d(o_w, o_{w+1}) \leq \varepsilon$ $(1 \leq w < n)$.

A **clustering result** $\mathcal{C}_k = \{c_1, c_2, \cdots, c_n\}$ is a set of clusters obtained from the snapshot $\mathcal{O}_k$. DBSCAN generates a clustering result according to a set of core points and their density reachable points [79].

2.3 Evolutionary Clustering

Evolutionary clustering performs clustering at each time step and evaluates the quality of clustering according to two aspects:

- Historical quality: how similar the current clustering $\mathcal{C}_k$ and the previous clustering $\mathcal{C}_{k-1}$ are;

- Snapshot quality: how similar the current clustering $\mathcal{C}_k$ and the clustering without smoothing $\mathcal{C}_o$ are.

Specifically, evolutionary clustering defines a historical cost $\mathcal{TC}_k$ and a snapshot cost $\mathcal{SC}_k$. The lower the cost, the higher the quality. Evolutionary clustering then integrates two costs into a cost function [72]:

$$\mathcal{F}_k = \mathcal{SC}_k(\mathcal{C}_o, \mathcal{C}_k) + \alpha \cdot \mathcal{TC}_k(\mathcal{C}_{k-1}, \mathcal{C}_k), \tag{4.1}$$

where α is a parameter enabling trade-offs between the two aspects. Generally, $\mathcal{C}_{opt}$ is the clustering result at dt_k that minimizes $\mathcal{F}_k$ [39, 68–76]. This suggests that a good clustering should evolve smoothly w.r.t. its previous counterpart and should be faithful to the current data. Existing studies generally apply iterative adjustements to achive clustering results $\mathcal{C}_{opt}$ that minimize Formula 4.1. Such iterative processes incur high time costs and thus are unsuitable for processing large-scale trajectories in real-time. An existing study [69] proposes *cost embedding* that pushes down the cost computation from the clustering result level to the data level. This technique enables efficient and flexible smoothing. We thus apply this strategy in ECO.

3 Problem Formulation

We introduce the new notions of snapshot cost and historical cost and formally define ECO.

3.1 Snapshot Cost

As we adopt cost embedding [69] that smooths trajectories at the data level, we need to define the location of a trajectory after smoothing.

Definition 3.1
An **adjustment** $r_k(o)$ is a location of a trajectory o obtained through smoothing at dt_k. Here, $r_k(o) \neq r_k(o')$ if $o \neq o'$. The set of adjustments in $\mathcal{O}_k$ is denoted as $\mathcal{R}_k$.

We simplify $r_k(o)$ to $r(o)$ if the context is clear. With Definition 3.1, we formulate the snapshot cost of a trajectory o as the distance between its location $o.l$ and its adjustment $r(o)$, which is in accordance with the idea of cost embedding:

$$\mathcal{SC}_k(r(o)) = d(r(o), o.l)^2 \quad s.t. \quad d(r(o), o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t}), \tag{4.2}$$

where μ is a speed constraint of the road network. Formula 4.2 constrains each adjustment to comply with the speed constraint.

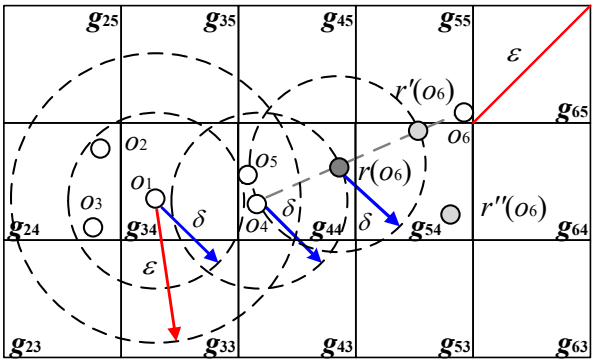


Figure 4.3: $o_i\,(1 \leq i \leq 6)$ at dt_2 in Figure 4.1 ($\rho = 3$) [67]

3.2 Historical Cost

Definition 3.2
A **seed point** $s\,(s \in \mathcal{S}_k)$ summarizes a **minimal group** $\mathcal{M}_k(s) = \{o \in \mathcal{O}_k | \, d(o,s) \leq \delta \wedge \forall s' \in \mathcal{S}_k\,(s' \neq s \Rightarrow d(o,s) \leq d(o,s'))\}$ at dt_k, where $\delta\,(0 < \delta \leq \varepsilon)$ is a given parameter and $\mathcal{S}_k\,(\mathcal{S}_k \subset \mathcal{O}_k)$ is a seed point set at dt_k. The cardinality of $\mathcal{M}_k(s)$ must exceed a parameter ρ. Any trajectory o in $\mathcal{M}_k(s)$ that is different from s is called a **non-seed point**. Note that, $\mathcal{M}_k(s) \cap \mathcal{M}_k(s') = \emptyset$ if $s \neq s'$.

Given the current time step dt_k, $\tilde{s}$ denotes the seed point of a non-seed point o at dt_{k-1} such that $o \in \mathcal{M}_{k-1}(\tilde{s})$.

Example 3.1
In Figure 4.2b, there are two minimal groups, i.e., $\mathcal{M}_1(o_1) = \{o_1, o_2, o_3\}$ and $\mathcal{M}_1(o_4) = \{o_4, o_5, o_6\}$. In Figure 4.3, there is only one minimal group before smoothing, i.e., $\mathcal{M}_2(o_1) = \{o_1, o_2, o_3\}$. Further, given the current $k = 2$, both s and $\tilde{s}$ of o_2 is o_1 and $\tilde{s}.\tilde{l} = o_1.l_1$.

Based on Definition 3.2, we formulate the temporal cost of a trajectory o as follows.

$$\begin{aligned} &\mathcal{TC}_k(r(o)) = \left(\lceil \frac{d(r(o), \tilde{s}.l)}{\delta} \rceil - 1 \right)^2 \\ &s.t.\, d(r(o), o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t}), \end{aligned} \tag{4.3}$$

where $o \in \mathcal{M}_{k-1}(\tilde{s}) \backslash \{\tilde{s}\}$. We use the *degree of closeness*, i.e., $\lceil \frac{d(r(o), \tilde{s}.l)}{\delta} \rceil$, to quantify the historical cost rather than using the exact distance $d(r(o), \tilde{s}.l)$, which is too strict considering the characteristics of trajectories.

3.3 Total Cost

The snapshot cost measures a distance, while the historical cost measures the degree of closeness. Thus, we first normalize them to the unit range.

$$\mathcal{SC}_k(r(o)) = \left(\frac{d(r(o), o.l)}{4\mu \cdot \Delta t + \delta}\right)^2 s.t.\, d(r(o), o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t}) \tag{4.4}$$

$$\mathcal{TC}_k(r(o)) = \left(\frac{\lceil \frac{d(r(o), \tilde{s}.l)}{\delta} \rceil - 1}{\frac{4\mu \cdot \Delta t + \delta}{\delta}}\right)^2 \quad s.t.\, d(r(o), o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t}), \tag{4.5}$$

where $o \in \mathcal{M}_{k-1}(\tilde{s}) \backslash \{\tilde{s}\}$ and Δt is the duration of a time step. Obviously, $\mathcal{SC}_k(r(o)) \geq 0$ and $\mathcal{TC}_k(r(o)) \geq 0$; thus, we only need to establish that $\mathcal{SC}_k(r(o)) < 1$ and $\mathcal{TC}_k(r(o)) < 1$.

Lemma 3.1
If $d(o.l, o.\tilde{l}) \leq (o.t - o.\tilde{t}) \cdot \mu$ then $d(r(o), o.l) \leq 4\mu \cdot \Delta t$.

Lemma 3.1 indicates that we must have $\mathcal{SC}_k(r(o)) < 1$ if $d(o.l, o.\tilde{l}) \leq (o.t - o.\tilde{t}) \cdot \mu$ always holds. In order to satisfy this constraint, we pre-process the location of a trajectory when it arrives, to be covered in Section 4.3.

Lemma 3.2
If $d(\tilde{s}.l, \tilde{s}.\tilde{l}) \leq (\tilde{s}.t - \tilde{s}.\tilde{t}) \cdot \mu$ then $d(r(o), \tilde{s}.l) \leq 4\mu \cdot \Delta t + \delta$.

We have $\lceil \frac{4\mu \cdot \Delta t + \delta}{\delta} \rceil - 1 < \frac{4\mu \cdot \Delta t + \delta}{\delta}$ and thus $\mathcal{TC}_k(r(o)) < 1$, based on Lemma 3.2. Setting $4\mu \cdot \Delta t + \delta = \pi$, the total cost $\mathcal{F}_k$ is:

$$\mathcal{F}_k = \sum_{o, \tilde{s} \in \Theta_k \wedge o \neq \tilde{s}} \frac{1}{\pi^2} \left(d(r(o), o.l)^2 + \alpha \cdot \left(\delta \cdot \left(\lceil \frac{d(r(o), \tilde{s}.l)}{\delta} \rceil - 1 \right) \right)^2 \right) \quad s.t.\, \forall o \in \Theta_k \, (d(r(o), o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t})), \tag{4.6}$$

where $\Theta_k = \mathcal{O}_k \cap (\bigcup_{\tilde{s} \in \mathcal{S}_{k-1}} \mathcal{M}_{k-1}(\tilde{s}))$. Based on Formula 4.6, we formulate the clustering problem as follows.

Definition 3.3
Given a snapshot $\mathcal{O}_k$, a set of previous minimal groups $\bigcup_{\tilde{s} \in \mathcal{S}_{k-1}} \mathcal{M}_{k-1}(\tilde{s})$; a time duration Δt; a speed constraint μ; and parameters α, δ, ε, *minPts*, and ρ, **evolutionary clustering of streaming trajectories** (**ECO**) is to

- finds a set of adjustments $\mathcal{R}_{k_{opt}}$ such that $\mathcal{R}_{k_{opt}} = \arg\min_{\mathcal{R}_k} \mathcal{F}_k$;
- computes a set of clusters $\mathcal{C}_k$ over $\mathcal{R}_{k_{opt}}$.

In particular, $r_{opt}(o)$ $(\in \mathcal{R}_{k_{opt}})$ denotes the adjustment of $o.l$ at dt_k and is then used as the previous location of o in the evolutionary clustering at dt_{k+1}. Existing solvers compute $\mathcal{R}_{k_{opt}}$ by iteratively adjusting each $r_{opt}(o) \in \mathcal{R}_{k_{opt}}$, as Formula 4.6 is neither continuous nor differentiable [80].

4 Computation of Adjustments

We proceed to present an approximate solution that enables minimizing the cost defined in Formula 4.6 in linear time. We also show the speed-based pre-processing that guarantees the correctness of the normalization introduced in the previous chapter.

4.1 Linear Time Solution

Formula 4.6 can be decomposed as follows.

$$\begin{aligned}\mathcal{F}_k &= \sum_{\tilde{s}\in\mathcal{S}_{k-1}} f_k(\tilde{s}.l) \\ &= \sum_{\tilde{s}\in\mathcal{S}_{k-1}} \sum_{o\in\Omega} \left(d(r(o), o.l)^2 + \alpha \cdot \left(\delta \cdot \left(\lceil \frac{d(r(o), \tilde{s}.l)}{\delta} \rceil - 1 \right) \right)^2 \right) \\ &= \sum_{\tilde{s}\in\mathcal{S}_{k-1}} \sum_{o\in\Omega} f_k(r(o), \tilde{s}.l) \\ s.t. \forall o \in \Theta_k \, &(d(r(o), o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t})),\end{aligned} \tag{4.7}$$

where $\Theta_k = \mathcal{O}_k \cap (\bigcup_{\tilde{s}\in\mathcal{S}_{k-1}} \mathcal{M}_{k-1}(\tilde{s}))$, $\Omega = \mathcal{M}_{k-1}(\tilde{s}) \backslash \{\tilde{s}\}$, $r(o)$ is the adjustment of $o.l$ at dt_k, $\tilde{s}$ is the seed point of o at dt_{k-1}, and $\tilde{s}.l$ is the location of $\tilde{s}$ at dt_k. We omit $\frac{1}{\pi^2}$ that is a constant.

Lemma 4.1
$\mathcal{F}_k$ achieves the minimum value if each $f_k(\tilde{s}.l)$ $(\tilde{s} \in \mathcal{S}_{k-1})$ achieves the minimum value.

Lemma 4.2
$f_k(\tilde{s}.l)$ achieves the minimum value if each $f_k(r(o), \tilde{s}.l)$ $(o \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k \backslash \{\tilde{s}\})$ achieves the minimum value.

Lemmas 4.1 and 4.2 suggest that $\mathcal{R}_{k_{opt}}$ can be obtained by computing each $r(o)$ $(o \in \Theta_k)$ that minimizes $f_k(r(o), \tilde{s}.l)$. We denote a **circle** as $\mathcal{Q}(e, x)$, where e is the center and x is the radius. Further, we denote a **segment** connecting two locations l and l' is denoted as $se(l, l')$. The **intersection** of a circle $\mathcal{Q}(e, x)$ and a segment $se(l, l')$ is denoted as $se(l, l') \oplus \mathcal{Q}(e, r)$.

Lemma 4.3
$se(o.l, \tilde{s}.l) \cap \mathcal{Q}(o.\tilde{l}, \mu \cdot (o.t - o.\tilde{t})) \neq \emptyset$.

Lemma 4.3 indicates that a feasible solution $r(o)$ on $se(o.l, \tilde{s}.l)$ must exist that satisfies the speed constraint in Formula 4.7.

Omitting the speed constraint. We first derive an adjustment $r_{opt'}(o)$ in linear time that minimizes $f_k(r(o), \tilde{s}.l)$ without considering the speed constraint. Then, we show how to compute $r_{opt}(o)$ approximately on the basis of $r_{opt'}(o)$.

Lemma 4.4
$\forall r'(o) \notin se(o.l, \tilde{s}.l)(\exists r(o) \in se(o.l, \tilde{s}.l)(f_k(r(o), \tilde{s}.l) \leq f_k(r'(o), \tilde{s}.l)))$.

Example 4.1
In Figure 4.3, $f(r(o_6), o_4.l) \leq f(r''(o_6), o_4.l)$ and $f(r'(o_6), o_4.l) \leq f(r''(o_6), o_4.l)$ due to $r''(o_6) \notin se(o_4.l, o_6.l)$.

Lemma 4.4 enables us to search $r_{opt'}(o)$ only on $se(o.l, \tilde{s}.l)$ without sacrificing accuracy.

Lemma 4.5
Let $d(r_{opt'}(o), \tilde{s}.l) = b_{opt'} \cdot \delta$. If $d(o.l, \tilde{s}.l) > \delta$ then $b_{opt'} \in \{\mathbf{N}^* \cap [\lambda_1, \lambda_2]\} \cup \lambda_2$, where $\lambda_1 = \frac{d(\tilde{s}.l, o.\tilde{l}) - \mu \cdot (o.t - o.\tilde{t})}{\delta}$, $\lambda_2 = \frac{d(o.l, \tilde{s}.l)}{\delta}$, and $\mathbf{N}^*$ is the natural numbers.

Lemma 4.5 further reduces the search space of $r_{opt'}(o)$ from a segment $se(o.l, \tilde{s}.l)$ to discrete points on the segment. According to Lemmas 4.3 to 4.5 and setting $d(r(o), \tilde{s}.l) = b \cdot \delta$, $f_k(r(o), \tilde{s}.l)$ can be transferred into the following function :

$$\begin{aligned} f_k(b, \tilde{s}.l) &= (d(o.l, \tilde{s}.l) - b \cdot \delta)^2 + \alpha \cdot (\delta \cdot (b-1))^2 \\ &s.t.\ b \in \{\mathbf{N}^* \cap [\lambda_1, \lambda_2]\} \cup \lambda_2, \end{aligned} \tag{4.8}$$

where $\lambda_1 = \frac{d(\tilde{s}.l, o.\tilde{l}) - \mu \cdot (o.t - o.\tilde{t})}{\delta}$ and $\lambda_2 = \frac{d(o.l, \tilde{s}.l)}{\delta}$. Clearly, the objective function of Formula 4.8 is continuous and differentiable, thus $b_{opt'}$ that minimizes $f_k(b, \tilde{s}.l)$ can be derived in constant time. As $d(r_{opt'}(o), \tilde{s}.l) = b_{opt'} \cdot \delta$ and $r_{opt'}(o) \in se(o.l, \tilde{s}.l)$ (cf. Lemma 4.4), $r_{opt'}(o)$ can be obtained directly by $b_{opt'}$.

Example 4.2
Continuing Example 3.2 and given $d(o_6.l, o_4.l) = 25$, $\alpha = 2.1$, and $\delta = 10$, we get $b_{opt'} = 1$ and $r_{opt'}(o_6) = r(o_6)$.

Introducing the speed constraint. We propose to efficiently compute r_{opt} by shrinking its feasible region from $\mathcal{Q}(o.\tilde{l}, \mu \cdot (o.t - o.\tilde{t}))$ to the search space of $r_{opt'}(o)$, which is a set of discrete points on $se(o.l, \tilde{s}.l)$. As Formula 4.8 is a simple quadratic function, b_{opt} can be derived approximately on top of $b_{opt'}$.

$$\begin{aligned} b_{opt} &= \arg \min_{b \in \{\mathbf{N}^* \cap [\lambda_1, \lambda_2]\} \cup \lambda_2} |b - b_{opt'}| \\ &s.t.\ \mathcal{Q}(o.\tilde{l}, \mu \cdot (o.t - o.\tilde{t})) \cap \mathcal{Q}(\tilde{s}.l, b \cdot \delta) \neq \emptyset \end{aligned} \tag{4.9}$$

Similar to $r_{opt'}(o)$, $r_{opt}(o)$ can be obtained directly after getting $b_{opt}(o)$. Intuitively, $r_{opt}(o) = r_{opt'}(o)$ if $r_{opt'}(o)$ satisfies the speed constraint in Formula 4.7. Empirical studies indicate that this happens in most cases. The underlying reasons are (i) that the speed of a trajectory generally does not exceed the speed constraint of a road network because of the limitations of road conditions and (ii) that the location of a trajectory has been adjusted for satisfying the speed constraint before smoothing (to be shown in Section 4.3). Overall, $r_{opt}(o)$ can be derived in constant time.

4.2 Shifting of Seed Points

In Section 4.1, we compute adjustments $o \in \mathcal{M}_{k-1}(\tilde{s}) \backslash \{\tilde{s}\}$ by assuming that the change from $\tilde{s}.l_{k-1}$ to $\tilde{s}.l_k$ is smooth, which is not always the case. This problem can be addressed by smoothing $\tilde{s}$ first. A previous study [80] cleans $\tilde{s}$ according to its close neighbors. Inspired by this, we use $o \in \mathcal{M}_{k-1}(\tilde{s}) \backslash \{\tilde{s}\}$ as a pivot to adjust $\tilde{s}$. However, according to Definition 3.2, $\mathcal{M}_{k-1}(\tilde{s}) \backslash \{\tilde{s}\}$ generally contains more than one non-seed point. This requires us to determine which non-seed point should be selected as the pivot to adjust $\tilde{s}$.

We consider a trajectory o to be normal if it moves smoothly, which is essentially evaluated by $f_k(b, \tilde{s}.l)$ in Formula 4.7. Thus, the pivot is chosen according to the following formula.

$$\tilde{s}_{new} = \arg \min_{o \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k} f_k(o) \tag{4.10}$$

Next, rather than using $\tilde{s}_{new}$ to smooth $\tilde{s}$ and then using $\tilde{s}$ to smooth $\mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k \backslash \{\tilde{s}\}$, we directly assign $\tilde{s}_{new}$ as the seed point for smoothing by Formula 4.7, i.e., we shift the seed point from $\tilde{s}$ to $\tilde{s}_{new}$. The underlying reasons are: (i) $\tilde{s}_{new}$ is identified as the trajectory with the smoothest movement among trajectories in $\mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k$ by Formula 4.10; thus, it is less necessary to smooth it; (ii)$|\mathcal{M}_{k-1}(\tilde{s})| - 1$ computations can be saved by using the seed point shifting strategy.

4.3 Speed-based Pre-processing

In order to ensure the correctness of the normalization of both snapshot and historical costs (cf. Formula 4.4), we adjust the locations of a trajectory as they arrive so that they satisfy the speed constraint, i.e., $d(o.l, o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t})$. We denote the location of a trajectory o before speed-based pre-processing as $o.l_o$ while that after speed-based pre-processing is still denoted as $o.l$.

According to the minimum change principle [20], the effect of the speed-based pre-processing on the subsequent smoothing should be as little as possible. Thus, we derive $o.l_o$, the non-seed points at the previous time step, according to the following formula.

$$\begin{aligned} &o.l = \arg \min_{o.l_p} \left| d(o.l_p, \tilde{s}.l) - d(o.l_o, \tilde{s}.l) \right| \\ &s.t.\, d(o.l_p - o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t}) \end{aligned} \tag{4.11}$$

Formula 4.11 implies that $o.l = o.l_o$ if $o.l_o$ satisfies the speed constraint. Since $\tilde{s}$ does not need to be smoothed by Formula 4.7, it is adjusted according to the following formula.

$$\begin{aligned} &\tilde{s}.l = \arg \min_{\tilde{s}.l_p} d(\tilde{s}.l_o, \tilde{s}.l_p) \\ &s.t.\, d(\tilde{s}.l_p - \tilde{s}.\tilde{l}) \leq \mu \cdot (\tilde{s}.t - \tilde{s}.\tilde{t}) \end{aligned} \tag{4.12}$$

Following the seed point shifting strategy, we check each $o \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k$ for identifying $\tilde{s}_{new}$. Hence, before this process, we need to ensure that each $o \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k$ follows the speed constraint w.r.t. the current to-be-examined seed point $\tilde{s}$ by Formulas 4.11 and 4.12.

5 Algorithm

Next, we present a grid index and the the algorithm underlying ECO.

5.1 Grid Index

We adopt a grid index [81] to accelerate the clustering algorithms. The diagonal of each grid cell (denoted as g) has length ε, which is one of the parameters used in DBSCAN [81]. An example is shown in Figure 4.3. We denote the smallest distance between the boundaries of two grid cells, g and g', as $\min(g, g')$. Based on this, we introduce the concept of h-closeness [81].

Definition 5.1
Two grid cells g and g' are ***h-close*** if $\min(g, g') \leq h$. The set of the ***h-close*** grid cells of g is denoted as $\mathcal{I}_h(g)$.

Lemma 5.1
For $o \in g$, we have $d(o.l, o'.l) > h$ if $o' \in g' \wedge g' \notin \mathcal{I}_h(g)$.

5.2 Generating Minimal Groups

The previous analysis suggests that we smooth a trajectory o only when it is contained in a minimal group. Since we hope to smooth as many trajectories as possible at each time step, the optimal set of minimal groups should satisfy $\forall o \in \mathcal{O}_k \backslash (\bigcup_{s \in \mathcal{S}_k} \mathcal{M}_k(s)) \forall s \in \mathcal{S}_k \ (d(o, s) > \delta))$. However, as we require $|\mathcal{M}_k(s)| \geq \rho$ and $d(o.l, s.l) \leq d(o.l, s'.l)$ if $o \in \mathcal{M}_k(s)$, we have to enumerate all the possible combinations to get the optimal set of $\mathcal{S}_k$. To make this feasible in streaming settings, we propose to greedily select a trajectory o as a seed point if $\forall s \in \mathcal{S}_k (d(o.l, s.l) > \delta$. Further, Lemma 5.1 enables us to search the seed point s of a non-seed point o only in $\mathcal{I}_\delta(o)$ according to Lemma 5.1 without sacrificing any accuracy.

5.3 Evolutionary Clustering

The overall ECO clustering algorithm first computes a set of adjustments $\mathcal{R}_{k_{opt}}$ according to Formulas 4.8–4.12. Then, it generates minimal groups according to $\mathcal{R}_{k_{opt}}$. Finally, it performs clustering on the basis of $\mathcal{R}_{k_{opt}}$. Further, we adapt an existing strategy [69] to optimizing modularity online and connect clusters in adjacent time steps following an existing proposal [69].

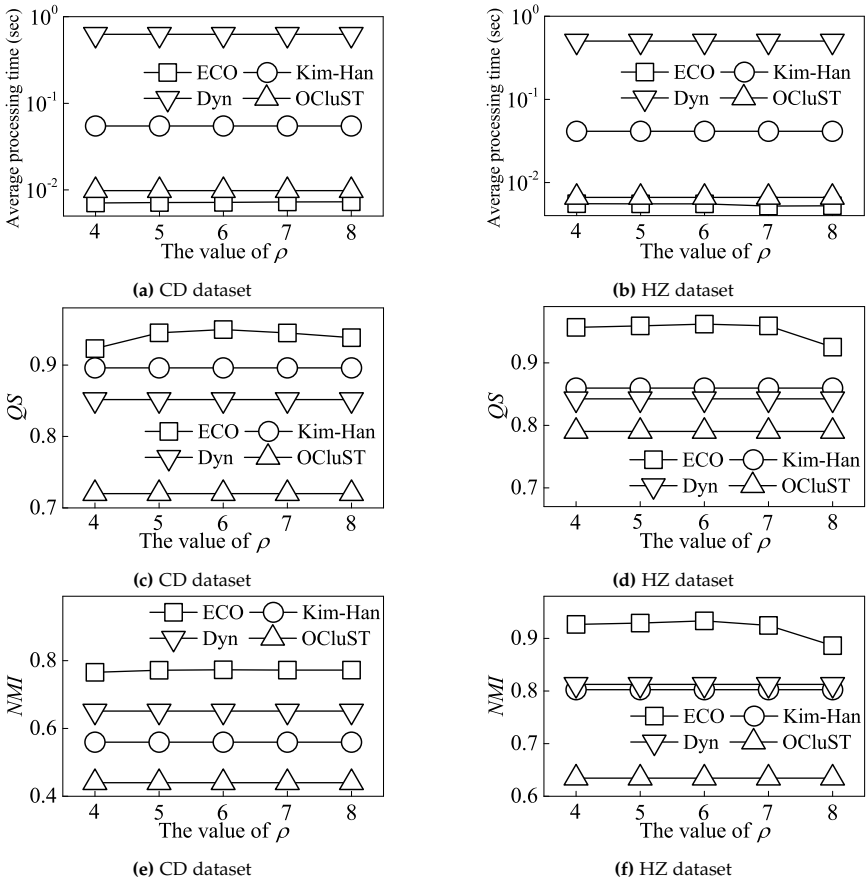


Figure 4.4: Comparison [67]

6 Experimental Evaluation

6.1 Experimental Design

Datasets. We use two real-life datasets, Chengdu (CD) and Hangzhou (HZ). The CD and HZ datasets contain 30 and 107 million GPS records, respectively.

Baselines and experimental settings. We use three baselines an existing proposal Kim-Han [69], DYN [74], and OCluST [29]. Kim-Han [69] is a representative density-based evolutionary clustering method. It improves efficiency by evaluating costs at the data level rather than at the cluster level. DYN [74] is the state-of-the-art evolutionary clustering technique. It adapts a particle swarm algorithm and random walks to improve clustering quality. OCluST [29] is the state-of-the-art for traditional clustering of streaming trajectories, but does not consider temporal smoothness. It constructs a novel structure to support continuous updates of representative trajectories in real-

time. All algorithms are implemented in C++, and the experiments are run on a computer with an Intel Core i9-9880H CPU (2.30 GHz) and 32 GB memory.

Performance metrics We use *modularity QS* to evaluate the clustering quality. A higher modularity indicates a better clustering. In addition, we use *normalized mutual information NMI* [82] to evaluate the smoothness of clustering over time. A higher *NMI* implies a more gradually change in the clustering. Finally, we use the *average processing time* per record at each time step to quantify efficiency.

6.2 Experimental Results

Comparison. Figure 4.4 reports the comparison results when varying ρ. First, ECO generally outperforms the baselines in terms of all performance metrics. The underlying reasons are: (i) ECO does not perform any iterations except for at the initial time step; (ii) ECO considers temporal smoothness and is designed specifically for trajectories, and (iii) ECO adjusts the location of a trajectory that does not move smoothly according to the location of a neighbor generally with high local density, increasing the intra-density and decreasing the inter-density of clustering.

Chapter 5

Conclusion and Future Work

1 Conclusion

Three papers are included in this thesis. Each paper formulates and solves a problem, as summarized below:

- Paper [14] proposes a framework, UTCQ, for compressing and querying uncertain trajectories in road networks. It presents an improved TED representation and a compact referential representation of uncertain trajectories in road networks. A SIAR scheme is proposed as a part of the improved TED representation for representing temporal information. To achieve high compression performance, a reference selection algorithm together with a Fine-grained Jaccard Distance are proposed, which enables selecting high-quality references efficiently. Further, a variable-length encoding scheme is presented for compressing referentially represented trajectories. In addition, an StIU index and efficient filtering and validation techniques are developed for partial decompression and efficient querying. Extensive experiments conducted using three real-life datasets show that the UTCQ framework outperforms the state-of-the-art solution in terms of both compression ratio and efficiency.
- Paper [52] proposes a framework, TRACE, for compressing and querying streaming trajectories in road networks. It presents a speed-based trajectory representation that exploits the similarity between sub-trajectories and a referential representation that uses multiple references. It adapts k-mer matching to trajectory representation and designs an online reference selection algorithm with an index structure to store references efficiently. Reference deletion and rewriting functions are also included in the framework. The former releases memory by detecting and deleting outdated references. The latter updates references according to the

most recent frequent patterns to improve compression. Moreover, a transmission strategy is developed that reduces the transmission cost while ensuring that compressed trajectories are decodable. A grid index and a filtering technique are also included in the framework to support real-time range queries. Extensive experiments conducted on two real-life datasets and one synthetic dataset show that TRACE outperforms the state-of-the-art solution in terms of both compression ratio and transmission cost.

- Paper [67] proposes a framework, ECO, for evolutionary clustering of streaming trajectories. Following existing studies, we adopt the idea of temporal smoothness and present new notions of so-called snapshot and historical costs for clusters of trajectories. To ensure that temporal smoothness is achieved efficiently, we develop a structure called minimal group, for summarizing neighboring trajectories. Next, we formulate the problem of evolutionary clustering of streaming trajectories, i.e., ECO. Then we formalize ECO as an optimization problem and prove that it can be solved approximately in linear time. A seed point shift strategy and a speed-based pre-processing are proposed along with the linear time solution. Finally, we present the algorithms that detail each component of ECO together with optimization techniques. Extensive experiments conducted on two real-life datasets show that ECO outperforms the state-of-the-art proposal in terms of both clustering quality and efficiency.

2 Future Work

In further research, it is of interest to adapt multiple-order representation to referential representation, which may further improve compression, and to enable query processing directly on compressed trajectories. It is also of interest to reduce the time delay and maximum memory cost for compression of streaming trajectories, without sacrificing the compression ratio and transmission cost. Moreover, deploying TRACE in a distributed cloud setting would be more practical and flexible for real-life applications. Finally, more information can be exploited for temporal smoothness in evolutionary clustering, which may enable improved performance.

Bibliography

References

[1] R. S. D. Sousa, A. Boukerche, and A. A. Loureiro, "Vehicle trajectory similarity: Models, methods, and applications," *CSUR*, vol. 53, no. 5, pp. 1–32, 2020.

[2] T. Cejka, V. Bartos, M. Svepes, Z. Rosa, and H. Kubatova, "Nemea: a framework for network traffic analysis," in *CNSM*. IEEE, 2016, pp. 195–201.

[3] J. S. C. Sheng Wang, Zhifeng Bao and G. Cong, "A survey on trajectory data management, analytics, and learning," *arXiv preprint arXiv:2003.11547*, 2020.

[4] Y. Zheng, "Trajectory data mining: an overview," *TIST*, vol. 6, no. 3, pp. 1–41, 2015.

[5] E. Cho, S. A. Myers, and J. Leskovec, "Friendship and mobility: user movement in location-based social networks," in *SIGKDD*, 2011, pp. 1082–1090.

[6] L. Liang, H. Ye, and G. Y. Li, "Toward intelligent vehicular networks: A machine learning framework," *IEEE Internet Things J.*, vol. 6, no. 1, pp. 124–135, 2018.

[7] P. Sun, S. Xia, G. Yuan, and D. Li, "An overview of moving object trajectory compression algorithms," *Math. Probl. Eng.*, vol. 2016, 2016.

[8] K. Zheng and H. Su, "Go beyond raw trajectory data: Quality and semantics." *IEEE Data Eng. Bull.*, vol. 38, no. 2, pp. 27–34, 2015.

[9] X. Zhang, L. Xie, Z. Wang, and J. Zhou, "Boosted trajectory calibration for traffic state estimation," in *ICDM*. IEEE, 2019, pp. 866–875.

[10] H. Su, K. Zheng, H. Wang, J. Huang, and X. Zhou, "Calibrating trajectory data for similarity-based analysis," in *SIGMOD*, 2013, pp. 833–844.

[11] D. Zhang, M. Ding, D. Yang, Y. Liu, J. Fan, and H. Shen, "Trajectory simplification: an experimental study and quality analysis," *PVLDB*, vol. 11, no. 9, pp. 934–946, 2018.

[12] M. Bierlaire, J. Chen, and J. Newman, "A probabilistic map matching method for smartphone GPS data," *TRANSPORT RES C-EMER*, vol. 26, pp. 78–98, 2013.

[13] S. Taguchi, S. Koide, and T. Yoshimura, "Online map matching with route prediction," *IEEE trans Intell Transp Syst*, vol. 20, no. 1, pp. 338–347, 2018.

[14] T. Li, R. Huang, L. Chen, C. S. Jensen, and T. B. Pedersen, "Compression of uncertain trajectories in road networks," *PVLDB*, vol. 13, no. 7, pp. 1050–1063, 2020.

[15] L.-Y. Wei, Y. Zheng, and W.-C. Peng, "Constructing popular routes from uncertain trajectories," in *KDD*, 2012, pp. 195–203.

[16] G. R. Jagadeesh and T. Srikanthan, "Probabilistic map matching of sparse and noisy smartphone location data," in *ITSC*, 2015, pp. 812–817.

[17] Y. Han, W. Sun, and B. Zheng, "Compress: A comprehensive framework of trajectory compression in road networks," *TODS*, vol. 42, no. 2, p. 11, 2017.

[18] Y. Ji, Y. Zang, W. Luo, X. Zhou, Y. Ding, and L. M. Ni, "Clockwise compression for trajectory data under road network constraints," in *ICBDA*, 2016, pp. 472–481.

[19] B. Krogh, C. S. Jensen, and K. Torp, "Efficient in-memory indexing of network-constrained trajectories," in *SIGSPATIAL*, 2016, pp. 17–26.

[20] R. Song, W. Sun, B. Zheng, and Y. Zheng, "Press: A novel framework of trajectory compression in road networks," *PVLDB*, vol. 7, no. 9, pp. 661–672, 2014.

[21] S. Koide, Y. Tadokoro, C. Xiao, and Y. Ishikawa, "CiNCT: Compression and retrieval for massive vehicular trajectories via relative movement labeling," in *ICDE*, 2018, pp. 1097–1108.

[22] X. Yang, B. Wang, K. Yang, C. Liu, and B. Zheng, "A novel representation and compression for queries on trajectories in road networks," *TKDE*, vol. 30, no. 4, pp. 613–629, 2017.

[23] A. Silva, R. Raghavendra, M. Srivatsa, and A. K. Singh, "Prediction-based online trajectory compression," *arXiv preprint arXiv:1601.06316*, 2016.

[24] J. Chen, Z. Xiao, D. Wang, D. Chen, V. Havyarimana, J. Bai, and H. Chen, "Toward opportunistic compression and transmission for private car trajectory data collection," *IEEE Sens. J.*, vol. 19, no. 5, pp. 1925–1935, 2018.

[25] G. Hu, J. Shao, F. Liu, Y. Wang, and H. Shen, "If-matching: Towards accurate map-matching with information fusion," *TKDE*, vol. 29, no. 1, pp. 114–127, 2016.

[26] W. Liu, Y. Zheng, S. Chawla, J. Yuan, and X. Xing, "Discovering spatio-temporal causal interactions in traffic data streams," in *SIGKDD*, 2011, pp. 1010–1018.

[27] Z. Chen, H. T. Shen, X. Zhou, Y. Zheng, and X. Xie, "Searching trajectories by locations: an efficiency study," in *SIGMOD*, 2010, pp. 255–266.

[28] F. I. Bashir, A. A. Khokhar, and D. Schonfeld, "Object trajectory-based activity classification and recognition using hidden markov models," *TIP*, vol. 16, no. 7, pp. 1912–1919, 2007.

[29] J. Mao, Q. Song, C. Jin, Z. Zhang, and A. Zhou, "Online clustering of streaming trajectories," *Front. Comput. Sci.*, vol. 12, no. 2, pp. 245–263, 2018.

[30] C. S. Jensen, D. Lin, and B. C. Ooi, "Continuous clustering of moving objects," *TKDE*, vol. 19, no. 9, pp. 1161–1174, 2007.

[31] Z. Li, J.-G. Lee, X. Li, and J. Han, "Incremental clustering for trajectories," in *DASFAA*. Springer, 2010, pp. 32–46.

[32] Y. Yu, Q. Wang, X. Wang, H. Wang, and J. He, "Online clustering for trajectory data stream of moving objects," *Comput. Sci. Inf. Syst.*, vol. 10, no. 3, pp. 1293–1317, 2013.

[33] G. Costa, G. Manco, and E. Masciari, "Dealing with trajectory streams by clustering and mathematical transforms," *Int. J. Intell. Syst.*, vol. 42, no. 1, pp. 155–177, 2014.

[34] Z. Deng, Y. Hu, M. Zhu, X. Huang, and B. Du, "A scalable and fast optics for clustering trajectory big data," *Cluster Comput*, vol. 18, no. 2, pp. 549–562, 2015.

[35] T. L. C. Da Silva, K. Zeitouni, and J. A. de Macêdo, "Online clustering of trajectory data stream," in *MDM*, vol. 1. IEEE, 2016, pp. 112–121.

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

[37] L.-A. Tang, Y. Zheng, J. Yuan, J. Han, A. Leung, C.-C. Hung, and W.-C. Peng, "On discovery of traveling companions from streaming trajectories," in *ICDE*. IEEE, 2012, pp. 186–197.

[38] X. Li, V. Ceikute, C. S. Jensen, and K.-L. Tan, "Effective online group discovery in trajectory databases," *TKDE*, vol. 25, no. 12, pp. 2752–2766, 2012.

[39] Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng, "Evolutionary spectral clustering by incorporating temporal smoothness," in *SIGKDD*, 2007, pp. 153–162.

[40] N. S. Nafi, R. H. Khan, J. Y. Khan, and M. Gregory, "A predictive road traffic management system based on vehicular ad-hoc network," in *ATNAC*. IEEE, 2014, pp. 135–140.

[41] V. Milanes, J. Villagrá, J. Godoy, J. Simó, J. Pérez, and E. Onieva, "An intelligent v2i-based traffic management system," *IEEE trans Intell Transp Syst*, vol. 13, no. 1, pp. 49–58, 2012.

[42] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *SIGMOD*, 2000, pp. 93–104.

[43] L. Li, X. Chen, Q. Liu, and Z. Bao, "A data-driven approach for gps trajectory data cleaning," in *DASFAA*. Springer, 2020, pp. 3–19.

[44] V. Patil, P. Singh, S. Parikh, and P. K. Atrey, "Geosclean: Secure cleaning of gps trajectory data using anomaly detection," in *MIPR*. IEEE, 2018, pp. 166–169.

[45] A. Idrissov and M. A. Nascimento, "A trajectory cleaning framework for trajectory clustering," in *MDC workshop*, 2012, pp. 18–19.

[46] M. Bierlaire and E. Frejinger, "Route choice modeling with network-free data," *TRANSPORT RES C-EMER*, vol. 16, no. 2, pp. 187–198, 2008.

[47] S. Deorowicz and S. Grabowski, "Robust relative compression of genomes with random access," *Bioinformatics*, vol. 27, no. 21, pp. 2979–2986, 2011.

[48] S. Wandelt and U. Leser, "Adaptive efficient compression of genomes," *ALGORITHM MOL BIOL*, vol. 7, no. 1, p. 30, 2012.

[49] W. Sebastian and L. Ulf, "Fresco: Referential compression of highly similar sequences," *TCBB*, vol. 10, no. 5, pp. 1275–1288, 2013.

[50] L. Chen, Y. Gao, X. Li, C. S. Jensen, and G. Chen, "Efficient metric indexing for similarity search," in *ICDE*, 2015, pp. 591–602.

[51] J. Teuhola, "A compression method for clustered bit-vectors," *INFORM PROCESS LETT*, vol. 7, no. 6, pp. 308–311, 1978.

[52] T. Li, L. Chen, C. S. Jensen, and T. B. Pedersen, "Trace: real-time compression of streaming trajectories in road networks," *PVLDB*, vol. 14, no. 7, pp. 1175–1187, 2021.

[53] P. Sui and X. Yang, "A privacy-preserving compression storage method for large trajectory data in road network," *J. Grid Comput.*, vol. 16, no. 2, pp. 229–245, 2018.

[54] 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 *KDD*, 2018, pp. 2797–2806.

[55] P. Zhao, Q. Zhao, C. Zhang, G. Su, Q. Zhang, and W. Rao, "Clean: frequent pattern-based trajectory spatial-temporal compression on road networks," in *MDM*, 2019, pp. 605–610.

[56] C. Chen, Y. Ding, Z. Wang, J. Zhao, B. Guo, and D. Zhang, "Vtracer: When online vehicle trajectory compression meets mobile edge computing," *IEEE Syst J*, vol. 14, no. 2, pp. 1635–1646, 2019.

[57] J. Muckell, J.-H. Hwang, V. Patil, C. T. Lawson, F. Ping, and S. Ravi, "Squish: an online approach for gps trajectory compression," in *COM.Geo*, 2011, pp. 1–8.

[58] Y. Li, W. Gao, W. Gao, H. Zhang, and J. Zhou, "A distributed double-newton descent algorithm for cooperative energy management of multiple energy bodies in energy internet," *IEEE T IND INFORM.*, 2020.

[59] C. Chen, Y. Ding, X. Xie, S. Zhang, Z. Wang, and L. Feng, "Trajcompressor: An online map-matching-based trajectory compression framework leveraging vehicle heading direction and change," *IEEE trans Intell Transp Syst*, vol. 21, no. 5, pp. 2012–2028, 2019.

[60] S. Deorowicz, "Fqsqueezer: k-mer-based compression of sequencing data," *Sci. Rep.*, vol. 10, no. 1, pp. 1–9, 2020.

[61] Y. Liu, H. Peng, L. Wong, and J. Li, "High-speed and high-ratio referential genome compression," *Bioinformatics*, vol. 33, no. 21, pp. 3364–3372, 2017.

[62] S. Saha and S. Rajasekaran, "Nrgc: a novel referential genome compression algorithm," *Bioinformatics*, vol. 32, no. 22, pp. 3405–3412, 2016.

[63] S. Raza, S. Wang, M. Ahmed, and M. R. Anwar, "A survey on vehicular edge computing: architecture, applications, technical issues, and future directions," *IEEE Wirel Commun*, vol. 19, no. 4, pp. 2322–2358, 2019.

[64] Y. Li, D. W. Gao, W. Gao, H. Zhang, and J. Zhou, "Double-mode energy management for multi-energy system via distributed dynamic event-triggered newton-raphson algorithm," *IEEE T Smart Grid.*, vol. 11, no. 6, pp. 5339–5356, 2020.

[65] Y. Chen and L. Tu, "Density-based clustering for real-time stream data," in *SIGKDD*, 2007, pp. 133–142.

[66] A. Dignös, M. H. Böhlen, and J. Gamper, "Overlap interval partition join," in *SIGMOD*, 2014, pp. 1459–1470.

[67] T. Li, L. Chen, C. S. Jensen, T. B. Pedersen, and J. Hu, "Evolutionary clustering of streaming trajectories," *arXiv preprint arXiv:2109.11609*, 2021.

[68] K. S. Xu, M. Kliger, and A. O. Hero III, "Adaptive evolutionary clustering," *Data Min Knowl Discov*, vol. 28, no. 2, pp. 304–336, 2014.

[69] M.-S. Kim and J. Han, "A particle-and-density based evolutionary clustering method for dynamic networks," *PVLDB*, vol. 2, no. 1, pp. 622–633, 2009.

[70] D. J. Fenn, M. A. Porter, M. McDonald, S. Williams, N. F. Johnson, and N. S. Jones, "Dynamic communities in multichannel data: An application to the foreign exchange market during the 2007–2008 credit crisis," *J Nonlinear Sci*, vol. 19, no. 3, p. 033119, 2009.

[71] J. Chen, C. Zhao, L. Chen *et al.*, "Collaborative filtering recommendation algorithm based on user correlation and evolutionary clustering," *Complex Syst.*, vol. 6, no. 1, pp. 147–156, 2020.

[72] D. Chakrabarti, R. Kumar, and A. Tomkins, "Evolutionary clustering," in *SIGKDD*, 2006, pp. 554–560.

[73] M. Gupta, C. C. Aggarwal, J. Han, and Y. Sun, "Evolutionary clustering and analysis of bibliographic networks," in *ASONAM*. IEEE, 2011, pp. 63–70.

[74] Y. Yin, Y. Zhao, H. Li, and X. Dong, "Multi-objective evolutionary clustering for large-scale dynamic community detection," *Inf. Sci.*, vol. 549, pp. 269–287, 2021.

[75] X. Ma and D. Dong, "Evolutionary nonnegative matrix factorization algorithms for community detection in dynamic networks," *TKDE*, vol. 29, no. 5, pp. 1045–1058, 2017.

[76] F. Liu, J. Wu, S. Xue, C. Zhou, J. Yang, and Q. Sheng, "Detecting the evolving community structure in dynamic social networks," *World Wide Web*, vol. 23, no. 2, pp. 715–733, 2020.

[77] F. Folino and C. Pizzuti, "An evolutionary multiobjective approach for community discovery in dynamic networks," *TKDE*, vol. 26, no. 8, pp. 1838–1852, 2013.

[78] F. Liu, J. Wu, C. Zhou, and J. Yang, "Evolutionary community detection in dynamic social networks," in *IJCNN*. IEEE, 2019, pp. 1–7.

[79] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *SIGKDD*, vol. 96, no. 34, 1996, pp. 226–231.

[80] S. Song, C. Li, and X. Zhang, "Turn waste into wealth: On simultaneous clustering and cleaning over dirty data," in *SIGKDD*, 2015, pp. 1115–1124.

[81] J. Gan and Y. Tao, "Dynamic density based clustering," in *SIGMOD*, 2017, pp. 1493–1507.

[82] A. Strehl and J. Ghosh, "Cluster ensembles—a knowledge reuse framework for combining multiple partitions," *J Mach Learn Res*, vol. 3, no. Dec, pp. 583–617, 2002.

Part II

Papers

Paper A

Compression of Uncertain Trajectories in Road Networks

Tianyi Li, Ruikai Huang, Lu Chen, Christian S. Jensen, and Torben Bach Pedersen

The paper has been published in the
International Conference on Very Large Data Bases (PVLDB), pp. 1050–1063, 2020.

The layout has been revised.

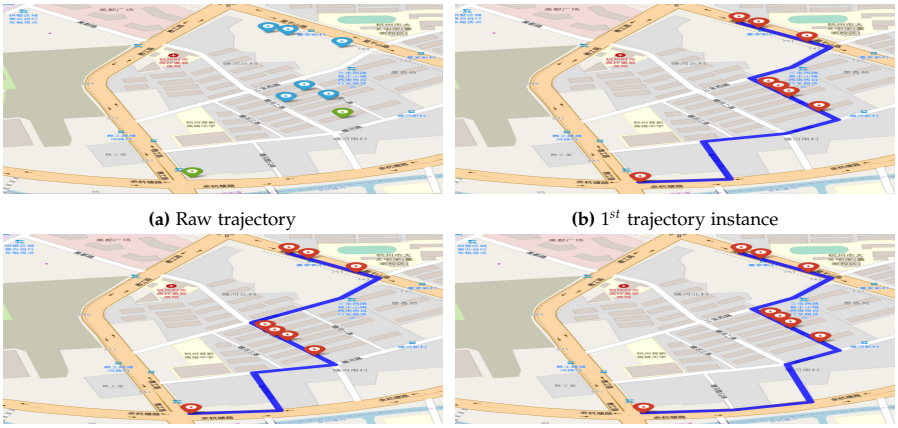
Abstract

Massive volumes of uncertain trajectory data are being generated by GPS devices. Due to the limitations of GPS data, these trajectories are generally uncertain. This state of affairs renders it is attractive to be able to compress uncertain trajectories and to be able to query the trajectories efficiently without the need for (full) decompression. Unlike existing studies that target accurate trajectories, we propose a framework that accommodates uncertain trajectories in road networks. To address the large cardinality of instances of a single uncertain trajectory, we exploit the similarity between uncertain trajectory instances and provide a referential representation. First, we propose a reference selection algorithm based on the notion of Fine-grained Jaccard Distance to efficiently select trajectory instances as references. Then we provide referential representations of the different types of information contained in trajectories to achieve high compression ratios. In particular, a new compression scheme for temporal information is presented to take into account variations in sample intervals. Finally, we propose an index and develop filtering techniques to support efficient queries over compressed uncertain trajectories. Extensive experiments with real-life datasets offer insight into the properties of the framework and suggest that it is capable of outperforming the existing state-of-the-art method in terms of both compression ratio and efficiency.

1 Introduction

GPS sensors in smart-phones, in-vehicle navigation systems, and wearable devices more generally are generating massive volumes of raw trajectories, increasing the cost of data storage and transmission [20, 21, 42]. This has led to the proposal of a variety of trajectory compression methods that fall into two general categories: raw trajectory compression and road network-embedded compression. The former category of methods aim to compress trajectories without considering an underlying road network [1, 3, 4, 6, 8, 9, 11, 13, 16, 22–25, 27, 29, 31, 40, 41, 44]. Methods in the latter category first project a trajectory onto a road network using a map-matching algorithm and then exploit the road network to reduce the storage by transforming the mapped trajectories into compact formats [5, 13, 17, 18, 31, 32]. As map-matching can help improve the quality of trajectories and as in-network trajectories are useful, we study road network-embedded compression.

The uncertainty of trajectories is caused by two characteristics, a low sampling rate and inaccurate GPS positions [38, 45]. A low sampling rate can make multiple routes between two map-matched GPS positions possible, while position inaccuracy means that a raw GPS position may be map-matched to multiple road-network positions. Here, a real-life example is shown in Fig. A.1a for a trajectory consisting of 8 raw GPS points recorded by a taxi



(a) Raw trajectory

(b) 1^{st} trajectory instance

(c) 2^{nd} trajectory instance

(d) 3^{rd} trajectory instance

Figure A.1: A real-life example of a raw trajectory and a corresponding network-constrained uncertain trajectory.

in Hangzhou, China. All these GPS points are off the road, and the sampling interval between two green points exceeds 4 minutes. To address this, probabilistic map-matching [2, 15] has been proposed to transform a raw trajectory into a set of network-constrained trajectory instances. Instead of mapping each position in a raw trajectory to a unique road-network location, probabilistic map-matching generally finds several potential road-network locations. Figs. A.1b, A.1c, and A.1d are trajectory instances generated from Fig. A.1a, which are similar to each other. To the best of our knowledge, no existing solutions aim to compress uncertain trajectories, which is the target of our study.

Two challenges must be addressed in order to effectively compress uncertain trajectories. *The first challenge is how to achieve a high compression ratio.* As shown in the example in Fig. A.1, the instances of an uncertain trajectory are similar, which is also validated by statistics from three real-life datasets, to be covered in Section 4.2. Hence, we adopt a referential representation for uncertain trajectories that has proven effective for highly similar genome sequences [10, 30, 35]. *The second challenge is how to support efficient querying of compressed uncertain trajectories.* An existing study [40] provides an index for accurate compressed trajectories that considers neither the uncertainty nor is applicable to referentially represented trajectory instances. Consequently, we propose a novel indexing technique that eliminates both shortcomings and also propose associated query processing algorithms.

We integrate these contributions into a novel framework that compresses uncertain trajectories and supports efficient query processing without the need for (full) decompression. First, we separate the temporal, path, and distance information of uncertain trajectories, thus following the state-of-

the-art TED model [40]. Here, we propose a representation of the temporal information to improve the compression ratio when sample intervals vary. Then, we represent trajectory instances referentially using a two-step process, i.e., selecting high-quality reference trajectory instances and transforming other trajectory instances accordingly. As part of this, we propose a Fine-grained Jaccard Distance function to measure the similarity between trajectories, and we propose a greedy algorithm to efficiently select high-quality references. Finally, we develop an index structure and algorithms that exploit effective filtering techniques and partial decompression to support typical probabilistic queries on compressed uncertain trajectories.

In summary, our main contributions are as follows:

- We propose a novel uncertain trajectory compression framework that supports efficient probabilistic query processing. To the best of our knowledge, this is the first proposal for compression of uncertain trajectories in road networks.
- We develop a representation that accommodates a novel encoding scheme for the temporal information of trajectories, and we use referential representation to compress uncertain trajectories in road networks.
- We design an effective indexing structure and filtering techniques to accelerate query processing, including the processing of *probabilistic where*, *when*, and *range queries*.
- We conduct extensive experimental evaluations that offer insight into the framework and demonstrate that it is able to outperform the state-of-the-art solution [40] by a factor of more than two in terms of the compression ratio and by more than one order of magnitude in terms of compression efficiency.

The rest of the paper is organized as follows. We present preliminaries in Section 2 and give an overview of the proposed framework in Section 3. Section 4 details the representation and compression schemes, and Section 6 covers the index structure and the query processing methodology. Section 6 reports the experimental results. Section 7 reviews related work, and Section 8 concludes and offers directions for future work.

2 Preliminaries

We proceed to introduce probabilistic map-matching and the TED model. Table B.1 summarizes frequently used notation.

Table A.1: Frequently used notation

Notation	Description
Tu	a set of uncertain trajectories
Tu^j	an uncertain trajectory in **Tu**
N^j	the number of instances in Tu^j
n_p^j	the number of pivots selected for Tu^j
Tu_w^j	an instance of the uncertain trajectory Tu^j
$SV(Tu_w^j)$	the start vertex of Tu_w^j
$D(Tu_w^j)$	the relative distance sequence of Tu_w^j
$Tu_w^j.p$	the probability of Tu_w^j
$E(Tu_w^j)$	the edge sequence of Tu_w^j
$T'(Tu_w^j)$	the time flag bit-string of Tu_w^j
$T(Tu^j)$	the time sequence of Tu^j
l_i	the i^{th} mapped location
Ref_i^j	the i^{th} reference in Tu^j
$Ref_i^j.Rrs$	the referential representation set of Ref_i^j
$Nref_{ik}^j$	the k^{th} non-reference in the set $Ref_i^j.Rrs$
$Com_\phi(Nref_{ik}^j, Ref_i^j)$	the referential representation of $Nref_{ik}^j$
$\phi_{ik}^j(Ma_h)$	the h^{th} factor in $Com_\phi(Nref_{ik}^j, Ref_i^j)$
$\hat{seq}$	the binary code of a sequence *seq*
ω_{seq}	the *flag array* of a sequence *seq*
γ_{seq}	the *original array* of a sequence *seq*

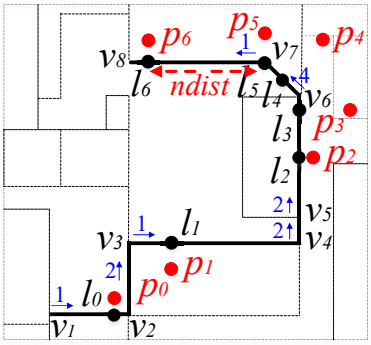
2.1 Probabilistic Map-Matching

A raw trajectory is a series of time-stamped point locations $p_0, ..., p_{n-1}$ of a moving object of the form (x, y, t), where (x, y) is a point location in 2D Euclidean space, with x being a longitude and y being a latitude, and t is a timestamp. $Tp = \langle p_0, ..., p_6 \rangle$ in Fig. A.2a is an example of a raw trajectory.

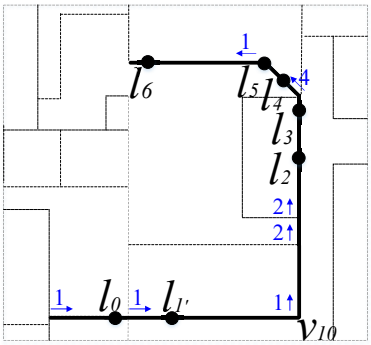
Definition 2.1
A *road network* is modeled as a directed graph $G = (V, E)$, where V is a set of vertices $v\,(x, y)$ denoting intersections or end points, and E is a set of directed edges $e\,(v_i \rightarrow v_j)$. Here, (x, y) denotes the 2D location of a vertex.

For simplicity, we use v and $(v_i \rightarrow v_j)$ to denote a vertex and a directed edge of a road network, respectively. **Map-matching** [14, 28] aligns a raw trajectory with the road network that constrains the movement of the corresponding object, and the result is a network-constrained accurate trajectory. This process transforms each original point location into a mapped location.

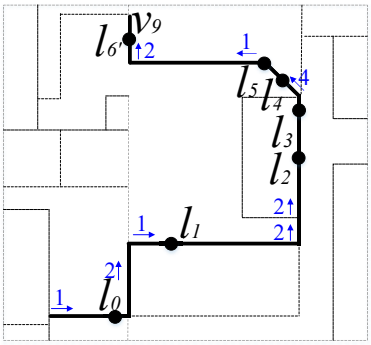
p_0=(120.0692, 30.28622, 5:03:25)
p_1=(120.0724, 30.28618, 5:07:25)
p_2=(120.0757, 30.28621, 5:11:26)
p_3=(120.0762, 30.28621, 5:15:26)
p_4=(120.0772, 30.28620, 5:19:25)
p_5=(120.0786, 30.28617, 5:23:25)
p_6=(120.0796, 30.28498, 5:27:25)



(a) Tu_1^1



(b) Tu_2^1



(c) Tu_3^1

Figure A.2: Instances of the network-constrained uncertain trajectory Tu^1

Definition 2.2
A *mapped location* l is a network-constrained location in a road network G, represented as $\langle (v_s \rightarrow v_e), ndist, t \rangle$, where $ndist$ is the network distance between v_s and l on $(v_s \rightarrow v_e)$ and t is a timestamp.

For example in Fig. A.2a, $l_6 = \langle (v_7 \rightarrow v_8), ndist, 5{:}27{:}25 \rangle$ is the mapped location corresponding to p_6; thus, the two have the same timestamp. We also denote a mapped location as $\langle (v_s \rightarrow v_e), ndist \rangle$ when the timestamp t is not considered.

Definition 2.3
A *network-constrained accurate trajectory* $Tr = \langle l_0, l_1, ..., l_{n-1} \rangle$ is a time-ordered sequence of mapped locations $l_0, l_1, ..., l_{n-1}$ of a moving object.

For example in Fig. A.2a, $Tr = \langle l_0, ..., l_6 \rangle$ is a network-constrained accurate trajectory.

Definition 2.4
Given two vertices v_s and v_e in a road network G, a *path* is a sequence of connected edges $(v_i \rightarrow v_j)$, that starts from v_s and ends at v_e, i.e., $(v_s \rightarrow v_1), (v_1 \rightarrow v_2), \cdots, (v_{n-1} \rightarrow v_e)$.

For example in Fig. A.2a, the path of Tp after map-matching is $(v_1 \rightarrow v_2), (v_2 \rightarrow v_3), \cdots, (v_7 \rightarrow v_8)$. A network-constrained **accurate** trajectory encodes a **unique** path of a raw trajectory after map-matching. However, to take into account the uncertainty in raw trajectories (as shown in Fig. A.1) when mapping them to the road network, **probabilistic map-matching** [2, 15] is proposed. Unlike a network-constrained accurate trajectory, a network-constrained **uncertain** trajectory encodes a set of **potential** paths, each of which is associated with a likelihood.

Definition 2.5
A *network-constrained uncertain trajectory* Tu^j contains a set of instances Tu^j_w $(1 \leq w \leq N^j)$ generated from a raw trajectory Tp. Each Tu^j_w is associated with a probability and is represented by a time-ordered sequence of mapped locations that is different from that of Tu^j_v $(1 \leq v \leq N^j \wedge v \neq w)$. All instances of Tu^j share the same temporal information for all mapped locations.

Fig. A.2 shows a network-constrained uncertain trajectory Tu^1 generated from the raw trajectory $Tp = \langle p_0, ..., p_6 \rangle$. Tu^1 contains three instances, i.e., Tu^1_1, Tu^1_2, and Tu^1_3, where $Tu^1_1 = \langle l_0, l_1, l_2, l_3, l_4, l_5, l_6 \rangle$ has probability 0.75, $Tu^1_2 = \langle l_0, l_{1'}, l_2, l_3, l_4, l_5, l_6 \rangle$ has probability 0.2, and $Tu^1_3 = \langle l_0, l_1, l_2, l_3, l_4, l_5, l_{6'} \rangle$ has probability 0.05. They share the temporal information ⟨5:03:25, 5:07:25, 5:11:26, 5:15:26, 5:19:25, 5:23:25, 5:27:25⟩, as they are all generated from Tp. In this example, the uncertain trajectory has three possible paths $(v_1 \rightarrow v_2), (v_2 \rightarrow v_3), (v_3 \rightarrow v_4), \cdots, (v_7 \rightarrow v_8)$ for Tu^1_1, $(v_1 \rightarrow v_2), (v_2 \rightarrow v_{10}), (v_{10} \rightarrow v_4), \cdots, (v_7 \rightarrow v_8)$ for Tu^1_2, and $(v_1 \rightarrow v_2), (v_2 \rightarrow v_3), (v_3 \rightarrow v_4), \cdots, (v_8 \rightarrow v_9)$ for Tu^1_3. In contrast, only Tu^1_1 that has the highest probability would be chosen as the network-constrained accurate trajectory Tr.

In the rest of the paper, we use accurate trajectory instead of network-constrained accurate trajectory, and we use uncertain trajectory instead of network-constrained uncertain trajectory when this does not cause any ambiguity.

2.2 TED Representation

TED represents an accurate trajectory Tr as an edge sequence $E(Tr)$, a time sequence $T(Tr)$, a time flag bit-string $T'(Tr)$, and a relative distance sequence $D(Tr)$. Fig. A.2a shows an example of Tr, where an object moves from $(v_1 \rightarrow v_2)$ to $(v_7 \rightarrow v_8)$ in sequence, and l_i $(0 \leq i \leq 6)$ are the mapped locations. The TED representation of Tr is shown in Table A.2.

Table A.2: An example of TED representation

$E(Tr)$	$\langle$185190 $\rightarrow$ 1,2,1,2,2,0,4,1,0$\rangle$
$D(Tr)$	$\langle$0.875,0.25,0.5,0.875,0.5,0,0.875$\rangle$
$T'(Tr)$	$\langle$1,0,1,0,1,1,1,1,1$\rangle$
$T(Tr)$	$\langle$(0, 5:03:25), (1, 5:07:25), (2, 5:11:26)
	(3, 5:15:26), (4, 5:19:25), (6, 5:27:25)$\rangle$

Edge Sequence. $E(Tr)$ is represented by a start vertex v followed by a sequence of outgoing edge numbers.

Definition 2.6
The outgoing edge number n_o (≥ 1) of an edge $(v_s \rightarrow v_e)$ means that $(v_s \rightarrow v_e)$ is the $n_o{}^{th}$ exit edge of v_s.

In Fig. A.2a, the edge sequence of Tr can be straightforwardly represented as $\langle(v_1 \rightarrow v_2), (v_2 \rightarrow v_3), ..., (v_7 \rightarrow v_8)\rangle$. Let ID of v_1 be 185190, and assume that $(v_1 \rightarrow v_2)$ is labeled as the first outgoing edge w.r.t. v_1. Then $(v_1 \rightarrow v_2)$ can be represented as 185190 $\rightarrow$ 1. Here, we keep the ID of v_1 to identify the start vertex of the path. Next, if $(v_2 \rightarrow v_3)$ is assigned as the second outgoing edge w.r.t. v_2, $(v_2 \rightarrow v_3)$ is encoded as 2. To capture that an edge $(v_s \rightarrow v_e)$ has r ($r > 1$) GPS points located on it, TED includes $(r-1)$ 0s after $(v_s \rightarrow v_e)$'s outgoing edge number w.r.t. v_s. As shown in Table A.2, the 0 after the 2 (2 is the outgoing edge number of $(v_5 \rightarrow v_6)$ w.r.t. v_5) in $E(Tr)$ indicates that $(v_5 \rightarrow v_6)$ has two mapped locations, i.e., l_2 and l_3.

Time Sequence. $T(Tr)$ is represented by omitting the consecutive timestamps with **unchanged** sample intervals. For example, $\langle t_i, t_{i+1}, t_{i+2}\rangle$ is encoded as $\langle(i, t_i), (i+2, t_{i+2})\rangle$ if $t_{i+2} - t_{i+1} = t_{i+1} - t_i$. However, statistics from real-life datasets show that the actual sample intervals vary frequently over time. To be specific, the sample interval changes every 6.80, 2.32 and 1.97 sample intervals on average on three real-life datasets, namely the Denmark (DK), Chengdu (CD), and Hangzhou (HZ) datasets, respectively. This translates into redundant representations and subsequent poor compression ratios for TED. We propose a new compression scheme that tackles this problem (Section 4.1).

Time Flag Bit-String. $T'(Tr)$ is a time flag bit-string that maps timestamps in $T(Tr)$ to outgoing edge numbers in $E(Tr)$. As shown in Table A.2, the mapping between the timestamps in $T(Tr)$ and outgoing edge numbers in $E(Tr)$ is not a one-to-one mapping. The reason is that an edge used by a trajectory may not contain any mapped location. For example, the fourth bit 0 of $T'(Tr)$ indicates that no GPS point is mapped to $(v_4 \rightarrow v_5)$; otherwise, it is set to 1.

Relative Distance Sequence. $D(Tr)$ is a sequence of relative distances of the mapped locations on their edges.

Definition 2.7
Given a mapped location $l = \langle (v_s \rightarrow v_e), ndist \rangle$, the *relative distance rd* of l w.r.t. $(v_s \rightarrow v_e)$ is the ratio of *ndist* to the length of $(v_s \rightarrow v_e)$ (denoted as $|(v_s \rightarrow v_e)|$).

For example in Fig. A.2a, *rd* of $l_6 = \langle (v_7 \rightarrow v_8), ndist \rangle$ w.r.t. $(v_7 \rightarrow v_8)$ is $\frac{ndist}{|(v_7 \rightarrow v_8)|}$. In the rest of the paper, we use $\langle (v_s \rightarrow v_e), ndist \rangle$ and $\langle (v_s \rightarrow v_e), rd \rangle$ to represent a mapped location l interchangeably, as they are semantically equivalent.

2.3 Compression with TED

We illustrate TED's compression [40] of $E(Tr)$, $T(Tr)$, $T'(Tr)$, and $D(Tr)$.

Compression on $E(Tr)$. Each $E(Tr)$ has a start vertex followed by a sequence of outgoing edge numbers, as illustrated in Section 2.2. TED compresses $E(Tr)$ using the following three steps: i) encoding each outgoing edge number using $\lceil \log_2(o) \rceil$ bits, where o is the maximal number of outgoing edges for all vertices $v \in V$; ii) grouping trajectories by the length of the binary code of $E(Tr)$, and forming an $A \times B$ binary code matrix for each group, where A is the number of trajectories and B is the length of $E(Tr)$; iii) applying multiple bases-based compression to each matrix, based on the observation that the highest bit of each code in the matrix has a high probability of being 0. Although the last two steps improve the compression ratio, we do not adopt them in our proposal due to two reasons. First, additional cost must be expended to group trajectories according to their lengths. Second, they require extra space for storing the auxiliary information and extra time for matrix operations during the multiple bases-based compression.

Compression on $T(Tr)$ and $T'(Tr)$. Each element in $T(Tr)$ is binary encoded while $T'(Tr)$ are bit-strings that are compressed using an existing bitmap compression algorithm [34].

Compression of $D(Tr)$. $D(Tr)$ is encoded by using a distance-preserving scheme. The binary code $C(rd)$ of a relative distance rd $(0 \leq rd < 1)$ is defined as $C(rd) = \sum_{i=0}^{I} C(rd_{x_i}) \frac{1}{2^i}$, where $C(rd_{x_i})$ is its i^{th} bit $(i \geq 0)$. Given an error bound η, I equals the smallest number of bits having $|C(rd) - rd| \leq \eta$. In addition, a PDDP-tree is proposed to further reduce the storage cost.

3 Framework

We study the compression and subsequent querying of network-constrained uncertain trajectories (NCUTs). We take $\mathbf{Tu} = \{Tu^j | 1 \leq j \leq M\}$, where M is the number of uncertain trajectories, as the input to our framework. Each Tu^j contains a set of instances Tu^j_w $(1 \leq w \leq N^j)$ consisting of i) $U_e(Tu^j_w)$

that is the edge sequence of Tu_w^j; ii) $U_d(Tu_w^j)$ that is the relative distances of all mapped locations; iii) $U_{t'}(Tu_w^j)$ that is the time flag bit-string to associate timestamps with edges; and iv) $U(Tu_w^j).p$ that is the probability associated with Tu_w^j. All Tu_w^j $(1 \leq w \leq N^j)$ share the same time sequence $U_t(Tu^j)$. Fig. A.3 depicts our framework for Uncertain Trajectory Compression and Querying (UTCQ) that encompasses three steps, where the red dashed box encloses the input and output to the steps, the blue dashed boxes capture the operations conducted during each step, and the orange dashed boxes capture techniques corresponding to each step.

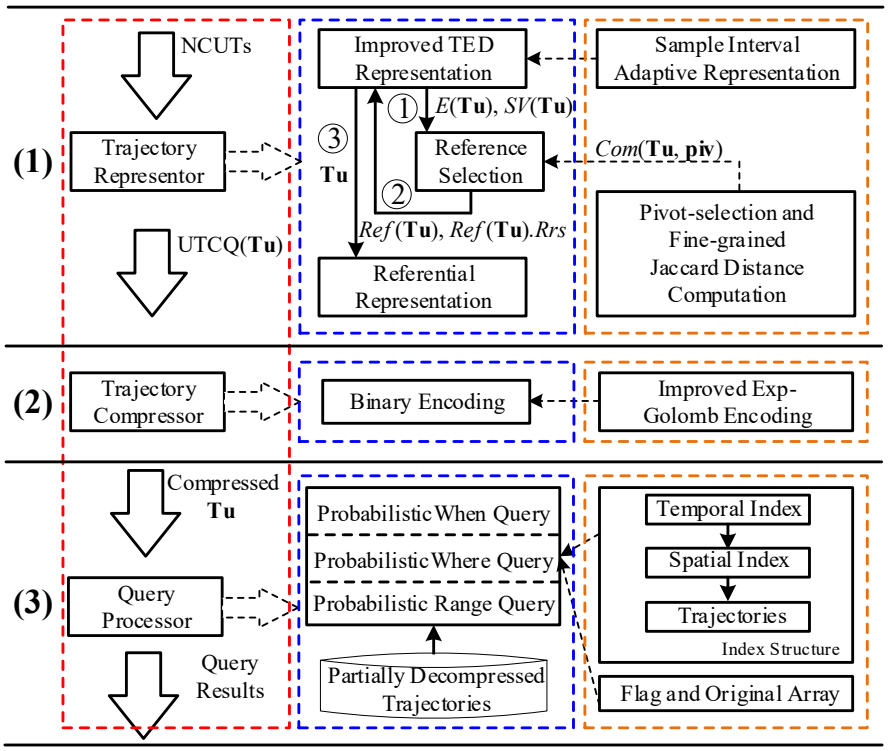


Figure A.3: Framework

The ***trajectory representor*** converts the NCUTs into a new format. This primarily involves three operations, i.e., improved TED representation, reference selection, and referential representation, as described below:

i) We separate the start vertices from the edge sequences $U_e(\mathbf{Tu}) = \{U_e(Tu^j) \mid Tu^j \in \mathbf{Tu}\}$ to obtain $SV(\mathbf{Tu})$ and $E(\mathbf{Tu})$, to achieve a more compact format. Moreover, we propose a new scheme to represent the time sequences $U_t(\mathbf{Tu}) = \{U_t(Tu^j) \mid Tu^j \in \mathbf{Tu}\}$ to achieve a high compression ratio. The details are presented in Section 4.1.

Table A.3: Example of improved TED representation of Tu^1 in Fig. A.2

w	1	2	3
$SV(Tu_w^1)$	185190	185190	185190
$E(Tu_w^1)$	⟨1, 2, 1, 2, 2, 0, 4, 1, 0⟩	⟨1, 1, 1, 2, 2, 0, 4, 1, 0⟩	⟨1, 2, 1, 2, 2, 0, 4, 1, 2⟩
$D(Tu_w^1)$	⟨0.875, 0.25, 0.5, 0.875 0.5, 0, 0.875⟩	⟨0.875, 0.25, 0.5, 0.875 0.5, 0, 0.875⟩	⟨0.875, 0.25, 0.5, 0.875 0.5, 0, 0.5⟩
$T'(Tu_w^1)$	⟨0, 1, 0, 1, 1, 1, 1⟩	⟨1, 0, 0, 1, 1, 1, 1⟩	⟨0, 1, 0, 1, 1, 1, 1⟩
$Tu_w^1.p$	0.75	0.2	0.05

ii) The representation of path information $SV(Tu^j)$ and $E(Tu^j)$ of each uncertain trajectory Tu^j is given to the *reference selector* (Fig. A.3①), that selects one or more high-quality reference instances for each uncertain trajectory. For each selected reference Ref_i^j, the *reference selector* finds a set of *non-references* (denoted as $Ref_i^j.Rrs$), i.e., other instances that can be represented by Ref_i^j. The reference selector sends the above results back to the improved TED representation step to identify the references as well as their non-references (Fig. A.3②). The detailed algorithm is covered in Section 4.3.

iii) For each uncertain trajectory Tu^j, the non-reference instances of Tu^j are represented according to their references, by applying a referential representation method (Fig. A.3③). Several formats are proposed to represent the non-references compactly. The details are provided in Section 4.2.

The ***trajectory compressor*** compresses the references and non-references into binary codes. Specifically, we adapt Exp-Golomb encoding [33] to compress $T(Tu^j)$, and we apply variable-length encoding to compress each non-reference based on its reference, in order to further reduce the storage cost. The detailed method is covered in Section 4.4.

The ***query processor*** is equipped with efficient algorithms that only partially decompress compressed trajectories. Specifically, we construct and use two auxiliary data structures to facilitate decompression of necessary information of non-references. In addition, we build a spatio-temporal index during compression that can effectively reduce the search space without the need for full decompression. The details are presented in Section 5.

4 Representor and Compressor

We present the expression and encoding scheme designed for uncertain trajectories. We use the running example depicted in Fig. A.2, which contains three instances (i.e., Tu_1^1, Tu_2^1, and Tu_3^1) of an uncertain trajectory Tu^1.

Table A.4: An referential representation example for Table 3

ϕ	$Com_\phi(Nref_{11}^1, Ref_1^1)$	$Com_\phi(Nref_{12}^1, Ref_1^1)$
SV	$\emptyset$	$\emptyset$
E	$\langle$(0, 1, 1), (2, 7)$\rangle$	$\langle$(0, 8, 2)$\rangle$
D	$\emptyset$	$\langle$(6, 0.5)$\rangle$
T'	$\langle$(1, 2), (3, 4)$\rangle$	$\emptyset$

4.1 Improved TED Representation

We denote each uncertain trajectory instance Tu_w^j $(1 \leq w \leq N^j)$ of a particular NCUT Tu^j as a tuple $(SV(Tu_w^j), E(Tu_w^j), D(Tu_w^j),$ $T'(Tu_w^j), Tu_w^j.p)$. Table A.3 shows how the example in Fig. A.2 is represented.

$\boldsymbol{SV(Tu_w^j)}$**.** $SV(Tu_w^j)$ is the start vertex ID of the first edge traversed by Tu_w^j. In Table A.3, $SV(Tu_1^1) = 185190$, which is the ID of v_1 in Fig. A.2a.

$\boldsymbol{D(Tu_w^j)}$ and $\boldsymbol{Tu_w^j.p}$**.** $D(Tu_w^j)$ and $Tu_w^j.p$ are the relative distance sequence (cf. Definition 2.7) and probability generated via the probabilistic map-matching process. As Tu^j consists of N^j instances, we have $\sum_{w=1}^{N^j} Tu_w^j.p = 1$.

$\boldsymbol{E(Tu_w^j)}$**.** $E(Tu_w^j)$ is the edge sequence of Tu_w^j (exclude the start vertex). We adopt the representation of edge sequence used by TED to represent it.

$\boldsymbol{T'(Tu_w^j)}$**.** We represent the time flag bit-string $T'(Tu_w^j)$ by modifying the TED representation slightly. The first and last edges traversed by Tu_w^j must each have at least one GPS point mapped onto them, so the first and last bits of $T'(Tu_w^j)$ must be 1. As a result, we omit the first and last bits when representing $T'(Tu_w^j)$, to improve the compression ratio.

The sample interval is unstable in real-life applications, and TED has a problem when representing such trajectories. In order to tackle this problem, we develop a new representation scheme to represent $T(Tu^j)$, namely Sample Interval Adaptive Representation (SIAR) .

Sample Interval Adaptive Representation of $\boldsymbol{T(Tu^j)}$. Fig. A.4a counts the differences between the actual sample intervals and the default ones using three real-life datasets, i.e., DK, CD, and HZ. As can be observed, most of the actual sample intervals (93% in DK, 62% in CD, and 54% in HZ) are equal to or deviate only 1 second from the default one. Motivated by this, we only record the *difference* between the (actual) sample interval and the default one. Assume that the time sequence of Tu^j starts with t_0 and that T_s is the default sample interval. SIAR keeps t_0 as the first value in $T(Tu^j)$ and represents the following timestamps as $(t_{i+1} - t_i) - T_s$, where t_i is the i^{th} timestamp. Given the time sequence $\langle$5:03:25, 5:07:25, 5:11:26, 5:15:26, 5:19:25, 5:23:25, 5:27:25$\rangle$, an example of SIAR used in UTCQ is $T(Tu^1) = \langle$5:03:25, 0, 1, 0, -1, 0, 0$\rangle$, where the default sample interval is 240 sec. In contrast, TED represents the sequence

as ⟨(0, 5:03:25), (1, 5:07:25), (2, 5:11:26), (3, 5:15:26), (4, 5:19:25), (6, 5:27:25)⟩, because most of its adjacent (actual) sample intervals are different. Hence, SIAR achieves a more compact representation when the sample interval varies frequently.

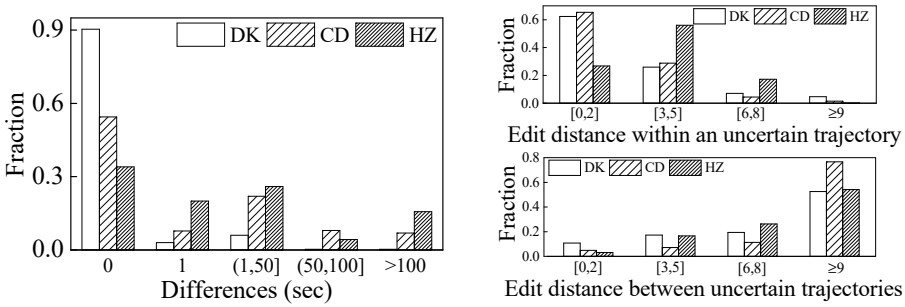
4.2 Referential Representation

The referential representation encodes the differences of an input sequence w.r.t. a reference sequence by exploiting the similarity between them (i.e., the more similar the sequences are, the higher the compression ratio). It is a **lossless** encoding [10, 30, 35]. Fig. A.4b shows statistics on the similarity between trajectory instances in DK, CD, and HZ. Here, we use edit distance to measure the similarity of $E(\cdot)$ between two instances as in [37, 43]. As can be observed, the edit distance between most of the instances of a particular uncertain trajectory (88% in DK, 94% in CD, and 83% in HZ) is at most 5, while that between most of the instances from different uncertain trajectories (53% in DK, 77% in CD, and 54% in HZ) is no less than 9. Hence, we only apply the referential representation to the trajectory instances of an uncertain trajectory rather than to the instances of different uncertain trajectories, to guarantee high compression ratios. For each uncertain trajectory, we select one or more instances as references, i.e., reference trajectory instances (see Section 4.3 for the selection). Then, other instances can be represented according to their reference using a set of *factors* defined below.

Definition 4.1
Given a non-reference $Nref_{ik}^{j}$ and its corresponding reference Ref_{i}^{j}, $Nref_{ik}^{j}$ can be expressed as a list of factors, i.e., $Com_{\phi}(Nref_{ik}^{j}, Ref_{i}^{j}) = \langle \phi_{ik}^{j}(Ma_h) | 1 \leq h \leq H \rangle$, where H is the number of factors, and a factor $\phi_{ik}^{j}(Ma_h)$ denotes a subsequence in $Nref_{ik}^{j}$.

Since one reference can be used for representing multiple non-references, we use the *referential representation set* $Ref_{i}^{j}.Rrs$ to denote the set of $Nref_{ik}^{j}$ ($1 \leq k \leq |Ref_{i}^{j}.Rrs|$) represented by Ref_{i}^{j}. Table A.4 shows the referential representation of Table 3, where Tu_1^1 is selected as the reference Ref_1^1 and is used to represent Tu_2^1 and Tu_3^1 (also called $Nref_{11}^1$ and $Nref_{12}^1$). The detailed representation is explained below.

***E*($Nref_{ik}^{j}$).** Several strategies exist for encoding a factor [10, 30, 35]. We adopt the (S, L, M) representation [30] to encode each factor of $Com_E(Nref_{ik}^{j}, Ref_{i}^{j})$, as it has a high compression ratio when the to-be-compressed sequence and the reference are highly similar. Specifically, S is the *start position* of the subsequence in the reference, L is the *length* of the subsequence, and M is the



(a) Sample intervals

(b) Similarity

Figure A.4: Statistics of real-life datasets

first *mis-matched element* following the subsequence. However, we rewrite the form of each factor in two cases:

A) When M does not exist (no mismatch), we record the factor as (S, L), omitting M. This does not introduce any ambiguity, since (S, L) only occurs at the end of the factor list.

B) When an outgoing edge number (denoted as n_o) in $E(Nref_{ik}^j)$ does not exist in $E(Ref_i^j)$, we denote the factor by assigning $S = |E(Ref_i^j)|$ and $M = n_o$, where $|E(Ref_i^j)|$ is the length of $E(Ref_i^j)$. The idea is to append n_o to the end of the reference, i.e., to consider n_o as its last value. We further omit L as it always equals 1. Hence, the factor in this case has the form (S, M). For example, given $E(Tu_4^1) = \langle 3, 2, 1, 2, 2 \rangle$, we have $E_{13}^1(Ma_1) = (9, 3)$, by referentially representing Tu_4^1 (denoted as $Nref_{13}^1$).

***T′*$(Nref_{ik}^j)$.** The referential representation of $T'(Nref_{ik}^j)$ is similar to that of $E(Nref_{ik}^j)$. We represent each factor using the format (S, L) because M can be inferred easily from the reference. To be specific, if the bit immediately following the longest prefix in $T'(Ref_i^j)$ is 1, the first mis-matched bit in $T'(Nref_{ik}^j)$ is 0, i.e., $M = 0$; otherwise, $M = 1$. Specifically, we always keep the last factor in the form (S, L, M) when M exists in order to avoid ambiguity. Then, if $T'(Nref_{ik}^j)$ is exactly the same as $T'(Ref_i^j)$, $Com_{T'}(Nref_{ik}^j, Ref_i^j) = \emptyset$ (as shown in Table A.4).

***D*$(Nref_{ik}^j)$.** We observe that even though a raw positional point could be mapped to two different edges via the probabilistic map-matching, the mapped locations may have the same relative distance, as the GPS records shown in Fig. A.1. Based on this observation, we encode each factor in $D(Nref_{ik}^j)$ using the format (pos, rd), where pos is the position of the different value rd. Note that this strategy is not good for referentially representing $E(Nref_{ik}^j)$ and $T'(Nref_{ik}^j)$ because their lengths vary across the instances of a single uncertain trajectory Tu^j.

Finally, we omit $SV(Nref_{ik}^{j})$. Because we do not choose a non-reference $Nref_{ik}^{j}$ that has a different start vertex than $Rref_{i}^{j}$. Also, we do not referentially compress $T(Nref_{ik}^{j})$ or $Nref_{ik}^{j}.p$, because i) all instances for a single uncertain trajectory Tu^{j} share the same $T(Tu^{j})$, and ii) the probability $Nref_{ik}^{j}.p$ has quite different values.

4.3 Reference Selection

Intuitively, the more similar a reference and a non-reference are, the higher the compression ratio. A naive reference selection strategy is to try every instance as the reference. However, its cost is too high. Inspired by an existing study [30], we use the similarity between the referential representations of trajectory instances to approximate their exact one. Here, we first select a set of *pivots*, and then represent each instance using these pivots. Thereafter, a *fine-grained Jaccard Distance function* is defined to estimate the similarity between two represented instances.

Pivot Selection. We select a set of pivots $\{piv_i | 1 \leq i \leq n_p^j\}$ from Tu^j. High-quality pivots are usually far away from each other and far away from other instances [7]. Hence, we i) randomly choose a trajectory instance and referentially represent all the remaining ones according to it; ii) select the one with the most factors as a pivot; iii) referentially represent all the trajectory instances using the most recently selected pivot; and iv) we repeat steps ii) and iii) until enough pivots are selected. Note that we only referentially represent $E(\cdot)$ of each trajectory instance by that of each pivot, as it is sufficient to distinguish the distances between instances.

Pivot Representation. We adopt the format (S, L) [10] to represent each factor in step iii). An example of (S, L) representation in Table A.3 is $Com_E(Tu_1^1, piv_1) = \langle (0, 8), (5, 1) \rangle$, where $piv_1 = Tu_3^1$. If an outgoing edge number in $E(Tu_w^j)$ does not exist in $E(piv_i)$, we omit the factor but increase the number of factors by 1. After pivot selection and representation, we get the referential representations of each trajectory Tu^j w.r.t. a set of pivots, i.e., $Com_E(Tu_w^j, piv_i)$ $(1 \leq w \leq N^j, 1 \leq i \leq n_p^j)$. The time complexity of pivot selection and representation for a Tu^j is $O(N^j \cdot n_p^j \cdot avg(|E|) \cdot avg(|Com_E|)$, where n_p^j is the number of pivots selected for Tu^j, $avg(|E|)$ is the average length of $E(\cdot)$ of all instances of Tu^j, and $avg(|Com_E|)$ is the average length of all instances w.r.t. all pivots of Tu^j.

Fine-grained Jaccard Distance Function. Given two trajectory instances Tu_w^j and Tu_v^j, and a pivot piv_i, we use the similarity between $Com_E(Tu_w^j, piv_i)$ and $Com_E(Tu_v^j, piv_i)$ to estimate the similarity between $E(Tu_w^j)$ and $E(Tu_v^j)$. Previous work [30] uses the Jaccard Distance to measure the simi-

larity. However, this distance is inaccurate in some cases. Given $piv_1 = Tu_3^1$ and $E(Tu_5^1) = \langle 1, 2, 1, 2, 2, 0, 4 \rangle$, we have $Com_E(Tu_1^1, piv_1) = \langle (0, 8), (5, 1) \rangle$ and $Com_E(Tu_5^1, piv_1) = \langle (0, 7) \rangle$. Thus, the Jaccard Distance between them is 1. However, $E(Tu_1^1)$ is actually very similar to $E(Tu_5^1)$. To obtain a more fine-grained distance notion, we propose a new distance metric, called *Fine-grained Jaccard Distance* (*FJD*), to calculate the distance from $E(Tu_w^j)$ to $E(Tu_v^j)$ against a pivot piv_i.

$$\begin{aligned} &FJD(Tu_w^j \rightarrow Tu_v^j, piv_i)(w \neq v) \\ &= \frac{\sum_{h'=1}^{H'} sim(E_{iv}^j(Ma_{h'}), Com_E(Tu_w^j, piv_i))}{\max\{H, H'\}}, \end{aligned} \tag{A.1}$$

where H and H' denote the number of factors in $Com_E(Tu_w^j, piv_i)$ and $Com_E(Tu_v^j, piv_i)$, respectively, while $E_{iv}^j(Ma_{h'})$ denotes the h'^{th} factor $(S_{h'}^{iv}, L_{h'}^{iv})$ in $Com_E(Tu_v^j, piv_i)$. In addition, we use $sim(E_{iv}^j(Ma_{h'}), Com_E(Tu_w^j, piv_i))$ to measure the similarity between $E_{iv}^j(Ma_{h'})$ and $Com_E(Tu_w^j, piv_i)$, calculated as follows,

$$\begin{aligned} &sim(E_{iv}^j(Ma_{h'}), Com_E(Tu_w^j, piv_i)) \\ &= \frac{\max_{h=1}^{H}(E_{iw}^j(Ma_h) \cap E_{iv}^j(Ma_{h'}))}{\max\{L_{\max}^{iw}, L_{h'}^{iv}\}} \end{aligned} \tag{A.2}$$

We define $E_{iw}^j(Ma_h) \cap E_{iv}^j(Ma_{h'})$ as $\max\{\min\{S_h^{iw} + L_h^{iw}, S_{h'}^{iv} + L_{h'}^{iv}\} - \max\{S_h^{iw}, S_{h'}^{iv}\}, 0\}$, and $L_{\max}^{iw} = \arg\max_{L_h^{iw}} E_{iw}^j(Ma_h) \cap E_{iv}^j(Ma_{h'})$. If $L_{\max}^{iw}$ is not unique, we choose the minimum value.

Example 4.1
(FJD computation) Consider the example in Table A.3. Given $piv_1 = Tu_3^1$, we have $Com_E(Tu_1^1, piv_1) = \langle (0, 8), (5, 1) \rangle$ and $Com_E(Tu_2^1, piv_1) = \langle (0, 1), (0, 1), (2, 6), (5, 1) \rangle$. Then we calculate $\frac{E_{11}^1(Ma_1) \cap E_{12}^1(Ma_1)}{\max\{L_1^{11}, L_1^{12}\}} = \frac{1}{8}$, in order to get $sim(E_{12}^1(Ma_1), Com_E(Tu_1^1, piv_1))$. Similarly, we are able to gain $sim(E_{12}^1(Ma_2), Com_E(Tu_1^1, piv_1)) = \frac{1}{8}$, $sim(E_{12}^1(Ma_3), Com_E(Tu_1^1, piv_1)) = \frac{3}{4}$, and $sim(E_{12}^1(Ma_4), Com_E(Tu_1^1, piv_1)) = 1$. Hence, $FJD(Tu_1^1 \rightarrow Tu_2^1, piv_1) = (\frac{1}{8} + \frac{1}{8} + \frac{3}{4} + 1)/4 = \frac{1}{2}$.

Based on $FJD(Tu_w^j \rightarrow Tu_v^j, piv_i)$, we present our score function $SF(Tu_w^j, Tu_v^j)$ $(w \neq v)$ for representing Tu_v^j by Tu_w^j, which is calculated as $Tu_w^j.p \cdot \max_{i=1}^{n_p^j} FJD(Tu_w^j \rightarrow Tu_v^j, piv_i)$. Here, a trajectory instance with higher probability of occurrence is expected to get a higher chance to be a reference, in order to speed up decompression during querying. Therefore, we multiply $Tu_w^j.p$ with the maximum *FJD* value. $SF(Tu_w^j, Tu_w^j)$ $(1 \leq w \leq N^j)$ is set to 0, as we

do not consider the case of representing a trajectory instance by itself. In addition, we calculate $SF(Tu_w^j, Tu_w^j)$ only when $SV(Tu_w^j) = SV(Tu_v^j)$, because two trajectory instances with different start vertices usually are not similar to each other. According to the score function SF, the optimal reference for Tu_v^j can be derived by the following formula:

$$Ref(Tu_v^j) = \arg\max_{Tu_w^j} SF(Tu_w^j, Tu_v^j) \tag{A.3}$$

There are two constraints in our setting: i) each non-reference only has one reference, to avoid redundancy; and ii) we only consider single-order compression. Our goal of reference selection is to maximize $\sum_{Tu_w^j, Tu_v^j \in Tu^j} SF(Tu_w^j, Tu_v^j)$ for an uncertain trajectory Tu^j under these two constraints. Unfortunately, we have to enumerate all the possible combinations to get the best selection choice, which is unfeasible, as the enumeration cost is $O(((N^j)^2)!)$ for Tu^j.

Therefore, we propose a greedy algorithm, shown in Algorithm 1, for selecting the references for each uncertain trajectory Tu^j $(1 \leq j \leq M)$. By applying SF to each pair of instances of an uncertain trajectory Tu^j, we can get a score matrix **SM**, where $SM[w][v] = SF(Tu_w^j, Tu_v^j)$ is the score of representing Tu_v^j by Tu_w^j. Algorithm 1 shows that we always choose the maximal element from **SM**, since it represents the current best reference. After each selection, we delete the elements in **SM** that do not satisfy the constraints (Line 7 and 9). The above-mentioned procedure is repeated until $\mathbf{SM} = \emptyset$ or the current maximum of it is 0. In the latter case, the trajectory instances that have not been selected are formally added to the reference set of Tu^j for easier query processing but are not associated with a reference representation set (Lines 11–13). The efficiency of Algorithm 1 can be further improved by presorting the elements in **SM**. The time complexity of reference selection for Tu^j is $O(N^j \cdot n_p^j \cdot avg(|E|) \cdot avg(|Com_E|) + {N^j}^2 \cdot n_p^j \cdot avg(|Com_E|)^2 + {N^j}^2 \cdot 2\log_2^{N^j})$, while the space complexity is $O(N^j \cdot n_p^j \cdot avg(|Com_E|) + (N^j)^2 \cdot n_p^j)$.

Example 4.2
(Algorithm 1 overview) Assuming that we only select Tu_3^1 as a pivot for Tu^1, we get an **SM**. Then we find the maximum in **SM**, i.e., $SF(Tu_1^1, Tu_2^1)$, based on which we get a reference Tu_1^1 and add Tu_2^1 to its Rrs. Afterwards, $SM[w'][2] \cup SM[2][w''] \cup SM[v'][1]$ $(1 \leq w', w'', v' \leq 3)$ are removed from **SM** according to the two constraints. This process is shown below,

$$\mathbf{SM} = \begin{bmatrix} 0 & \frac{3}{8} & \frac{1}{3} \\ \frac{7}{80} & 0 & \frac{1}{30} \\ \frac{1}{40} & \frac{1}{80} & 0 \end{bmatrix} \rightarrow \begin{bmatrix} \cancel{0} & \cancel{\frac{3}{8}} & \frac{1}{3} \\ \cancel{\frac{7}{80}} & \cancel{0} & \cancel{\frac{1}{30}} \\ \cancel{\frac{1}{40}} & \cancel{\frac{1}{80}} & 0 \end{bmatrix}$$

Then we add Tu_3^1 to $Tu_1^1.Rrs$ due to $SM[1][3] > SM[3][3]$, and remove $SM[w'][3]$

Algorithm 1: Reference Selection Algorithm

Input: **SM** of Tu^j
Output: the reference set $Ref(Tu^j)$ of Tu^j, and the referential representation set of each reference in $Ref(Tu^j)$ if it exists

initialize an empty reference set $Ref(Tu^j)$
while **SM** $\neq \emptyset$ **do**
  select the maximum score $SM[w][v]$ from **SM**
  if $SM[w][v] > 0$ **then**
    if $Tu^j_w \notin Ref(Tu^j)$ **then**
      add Tu^j_w to $Ref(Tu^j)$, and create $Tu^j_w.Rrs$
      remove $SM[v'][w]$ from **SM** $(1 \leq v' \leq N^j)$
    add Tu^j_v to $Tu^j_w.Rrs$
    remove $SM[w'][v]$, $SM[v][w'']$ from **SM** $(1 \leq w', w'' \leq N^j)$
  else
    for *each diagonal element* $SM[w][w]$ *in* **SM** **do**
      if $SM[w][w]$ *exists* **then**
        add Tu^j_w to $Ref(Tu^j)$
return $Ref(Tu^j)$ and the non-empty $Tu^j_w.Rrs$ for each $Tu^j_w \in Ref(Tu^j)$

$\cup SM[3][w''](1 \leq w', w'' \leq 3)$ from **SM**. Finally, since **SM** $= \emptyset$, we return the reference Tu^1_1 with its $Rrs = \{Tu^1_2, Tu^1_3\}$ for Tu^1.

4.4 Compression

Binary Encoding References. We follow TED [40] to compress $E(Ref)$. As discussed in Section 2.2, we omit the time-consuming procedures of TED. In this case, UTCQ still outperforms TED in terms of compressing ratio due to the referential representation and compression, while significantly improving the compression efficiency. Also, we adopt the PDDP-tree [40], which is the only lossy component in our framework, to encode $D(Ref)$ and $Ref.p$ that are floats. We use the error bounds η_D and η_p to constrain their compression accuracy. Both η_D and η_p are pre-set compression parameters, and the actual errors between the original and compressed data are constrained by these settings. $T'(Ref)$ is already represented as bit strings as discussed in Section 4.1 and does not need any further compression. Moreover, we propose an improved Exp-Golomb encoding to compress $T(Ref)$, which effectively addresses the sample interval fluctuation.

Improved Exp-Golomb Encoding. As different deviations between the actual sample interval and the default occur with different frequencies, encod-

ing each value of $T(Tu^j)$ in a binary code with fixed length may waste space. Specifically, we find that small deviations are much more frequent than large ones. The statistics in Fig. A.4a exemplify this. Thus, we adopt the well-known Exp-Golomb encoding [33] to compress $T(Tu^j)$. It encodes smaller values with shorter lengths and larger values with longer lengths. We set the parameter k, which controls the length of the first group, to 0. Given timestamps t_i and t_{i+1} of an uncertain trajectory Tu^j with default sample interval T_s, we let $\triangle t_i = (t_{i+1} - t_i) - T_s$ $(0 \leq i < |T(Tu^j)| - 1)$, where $|T(Tu^j)|$ is the length of $T(Tu^j)$. However, since $\triangle t_i$ may be negative, the Exp-Golomb encoding needs to be modified.

Assuming that the longest actual sample interval is T_l, we have $\triangle t_i \in (-T_s, T_l - T_s]$ $(0 \leq i < |T(Tu^j)| - 1)$. We divide $[0, \max\{T_s - 1, T_l - T_s\}]$ into n groups, where $n = \lceil \log_2(\max\{T_s - 1,\ T_l - T_s\} + 1) \rceil$, and the range of the $j^{th}(j \geq 0)$ group is $\left[-2^{j+1} + 2,\ \ -2^j + 1\right] \cup \left[2^j - 1, 2^{j+1} - 2\right]$. This way, all the possible deviations between the actual sample interval and the default one can be covered as $[-\max\{T_s - 1, T_l - T_s\}, \max\{T_s - 1, T_l - T_s\}] \subseteq [-2^n + 2, 2^n - 2]$. The offset of $\triangle t_i$ in the j^{th} group is given by $|\triangle t_i| - (2^j - 1)$. Moreover, we add one 1 bit immediately before the offset if $\triangle t_i$ is a negative digit; otherwise, 0 is added. Following the example of SIAR in Section 4.1, $\langle$5:03:25, 0, 1, 0, -1, 0, 0$\rangle$ is encoded as $\langle$00100011100011101, 0, 1000, 0, 1010, 0, 0$\rangle$ by the improved Exp-Golomb encoding. Hence the compression ratio of $T(Tu^1)$ by our method is $\frac{32\times7}{12+17} = 7.72$, while the counterpart by TED is $\frac{32\times7}{(17+12)\times6} = 1.29$, where we assume each trajectory contains at most 2^{12} timestamps and t_i $(0 \leq i < |T(Tu^j)|)$ is encoded using 17 bits.

Binary Encoding Non-references. Let $|E(Ref_i^j)|$ be the length of $E(Ref_i^j)$ and o be the maximum number of outgoing edges for any vertex $v \in V$. Then S takes $\lceil \log_2 |E(Ref_i^j)| + 1 \rceil$ bits, L takes $\lceil \log_2 |E(Ref_i^j)| \rceil$ bits, and M takes $\lceil \log_2 o \rceil$ bits when performing binary encoding of a factor (S, L, M) in $E(Nref_{ik}^j)$. Next, S and L in the factor of $T'(Nref_{ik}^j)$ are encoded in $\lceil \log_2 |T'(Ref_i^j)| \rceil$ bits, while M takes 1 bit. Further, *pos* in each factor of $D(Nref_{ik}^j)$ occupies $\lceil \log_2 |D(Ref_{ik}^j)| \rceil$ bits, and *rd* is encoded by a PDDP-tree [40]. It can be seen that, by using referential representation, binary codes of different non-references may have different lengths depending on their similarities to the corresponding references, further saving space. We denote the binary code of a sequence *seq* as *sêq* in the rest of the paper, e.g., the binary code of $E(\cdot)$ is denoted as $\hat{E}(\cdot)$.

Overall, the space complexity of compressing Tu^j is $O(N^j \cdot n_p^j \cdot avg(|Com_E|) + (N^j)^2 \cdot n_p^j + size_{in}(Tu^j) + size_{out}(Tu^j))$, where $size_{in}(Tu^j)$ and $size_{out}(Tu^j)$ are the input size and the compressed size of Tu^j, respectively.

5 Query Processor

In this section, we describe how to compute queries directly on compressed uncertain trajectories. First of all, we introduce a strategy to extract the necessary information from compressed *time flag bit-strings* by means of partial decompression. Second, we design an index to achieve fast retrieval and partial decompression. In addition, several pruning techniques are proposed to more efficiently support probabilistic queries on compressed uncertain trajectories.

5.1 Time Flag Bit-string Decompression

We have represented the time flag bit-strings of non-references as a list of factors. Since time flag bit-strings associate $D(\cdot)$ and $T(\cdot)$ with $E(\cdot)$, we need to get the number of 1s before any position in order to support queries over compressed data [40]. Naively, we can first decompress factors of $\hat{Com}_{T'}(Nref_{ik}^j, Ref_i^j)$ to $T'(Nref_{ik}^j)$, and then count the number, with the cost $O(|\hat{Com}_{T'}(Nref_{ik}^j, Ref_i^j)| + |T'(Nref_{ik}^j)|)$. To accelerate this, we propose an effective method by constructing two assisting arrays, *flag array and original array*.

Flag Array and Original Array. The *flag array* of $T'(Ref_i^j)$ is denoted as $\omega_{T'(Ref_i^j)}$, and counts the number of 1s before the g^{th} (not including g) bit of $T'(Ref_i^j)$ ($0 < g \leq |T'(Ref_i^j)|$). However, $T'(Ref_i^j)$ omits the first and last bit of the original time flag bit-string during representation (in Section 4.1). Therefore, we define the *original array*, $\gamma_{T'(Ref_i^j)}$, which records the number of 1s until the g^{th} bit in the original time flag bit-string ($0 \leq g < |T'(Ref_i^j)|$). Here, we simplify $\varphi_{T'(Ref_i^j)}$ and $\varphi_{T'(Nref_{ik}^j)}$ by representing them as $\varphi_{Ref_i^j}$ and $\varphi_{Nref_{ik}^j}$, where $\varphi \in \{\omega, \gamma\}$.

For a reference Ref_i^j, it is easy to get $\omega_{Ref_i^j}$ by linearly scanning $T'(Ref_i^j)$. To get $\gamma_{Nref_{ik}^j}$ for a non-reference, we propose a strategy by partially decompressing $Com_{T'}(Nref_{ik}^j, Ref_i^j)$. Given $\omega_{Ref_i^j}$ and g, we can get $\gamma_{Nref_{ik}^j}[g]$ by only decompressing at most one factor in $Com_{T'}(Nref_{ik}^j, Ref_i^j)$. More specifically, we first locate the factor in $Com_{T'}(Nref_{ik}^j, Ref_i^j)$ that the g^{th} bit of the original $T'(Nref_{ik}^j)$ falls into, as follows.

$$\max h \quad s.t.\, h + \sum_{l=1}^{h} L_l^{ik} \leq g \wedge h < H, \tag{A.4}$$

where H is the number of factors in $Com_{T'}(Nref_{ik}^j, Ref_i^j)$ and L_l^{ik} is the length of

the subsequence represented by ${T'}^j_{ik}(Ma_l)$. Formula A.4 ensures that the g^{th} bit of the original $T'(Nref^j_{ik})$ either falls into ${T'}^j_{ik}(Ma_{h+1})$ or exactly corresponds to the M that is omitted in ${T'}^j_{ik}(Ma_h)$. Thus we only need to decompress ${T'}^j_{ik}(Ma_{h+1})$ after calculating the number of 1s within the subsequence before the $(h+1)^{th}$ factor, as follows.

$$\begin{aligned} Z =& 1 + \sum_{l=1}^{h} \omega_{Ref_i^j}[S_l^{ik} + L_l^{ik}] - \omega_{Ref_i^j}[S_l^{ik}] \\ &+ \sim T'(Ref_i^j)[S_l^{ik} + L_l^{ik}], \end{aligned} \tag{A.5}$$

where $\sim (x)$ means NOT(x), i.e., the neglected mismatched elements of ${T'}^j_{ik}(Ma_l)$, and S_l^{ik} refers to the start position of the subsequence represented by ${T'}^j_{ik}(Ma_l)$. Let $g' = g - h - \sum_{l=1}^{h} L_l^{ik}$. Then $\gamma_{Nref^j_{ik}}[g]$ can be derived as follows.

$$\gamma_{Nref^j_{ik}}[g] = Z + \omega_{Ref_i^j}[S_{h+1}^{ik} + g'] - \omega_{Ref_i^j}[S_{h+1}^{ik}], \tag{A.6}$$

where $g \geq L_1^{ik} + 1 \wedge g < H + \sum_{l=1}^{H} L_l^{ik} (H \neq 1)$.

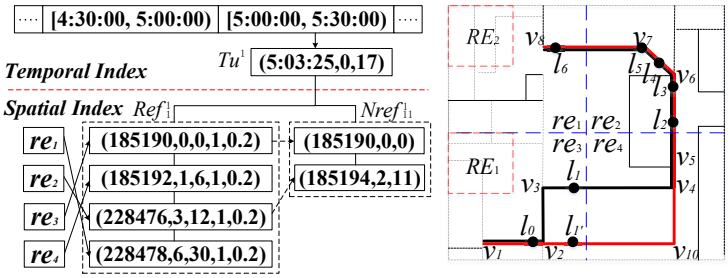
Hence, we extract the necessary information $\gamma_{Nref^j_{ik}}[g]$ with the cost $O(|\hat{Com}_{T'}(Nref^j_{ik}, Ref^j_i)| + \frac{|T'(Nref^j_{ik})|}{H})$ if g is given, where $\frac{|T'(Nref^j_{ik})|}{H}$ is the average length of each factor.

5.2 StIU Index

We propose an index, called Spatio-temporal Information based Uncertain Trajectory Index (StIU), to support efficient probabilistic queries by partial decompression. The partial decompression is lossless and decompresses only the information necessary for answering queries [40]. As shown in Fig. A.5a, an StIU index is built on Tu^1, where Tu_1^1 (used as Ref_1^1) and Tu_2^1 (used as $Nref_{11}^1$) are instances of the uncertain trajectory Tu^1 depicted in Fig. A.2. Let the IDs of v_1, v_2, v_3, v_4, v_5, and v_7 be 185190, 185191, 185192, 185194, 228476, and 228478, respectively. Assume that the mapping positions of the relative distances of l_1, l_2, and l_5 in $\hat{D}(Ref_1^1)$ are 6, 12, and 30, respectively, and that the maximum outgoing edge number of the road network in Fig. A.5b is 7. The index contains two parts. The upper part indexes the temporal information of trajectories, while the lower part supports effective spatial search.

Temporal Index of StIU. We first partition a day into equal-length time intervals. Then, we associate each interval with the uncertain trajectories, whose timestamps intersect with it. The information on an uncertain trajectory Tu^j corresponding to a time interval is stored in a tuple (*t.start, t.no, t.pos*), where *t.start* is the earliest timestamp of Tu^j falling into the time interval, *t.no*

indicates that *t.start* is the $t.no^{th}$ timestamp in $T(Tu^j)$, and *t.pos* refers to the matching position of the $(t.no+1)^{th}$ timestamp in $\hat{T}(Tu^j)$.



(a) Index Structure **(b)** Tu_1^1 and Tu_2^1

Figure A.5: StIU built on $\boldsymbol{Tu} = \{Tu^1\}$ depicted in Fig. A.2

Spatial Index of StIU. We further organize the trajectory instances in each time interval according to their spatial information. To be specific, we first partition the road network G using grid cells, each of which represents a region re_i. Then we create tuples, each linking a trajectory instance to a region it has passed. The tuples for a trajectory instance form a chronologically ordered list. For example, in Fig. A.5a, the tuple associated with re_3 is followed by that associated with re_4 for Tu_1^1 (used as Ref_1^1). Before detailing the index information stored in each tuple of Tu_w^j, we introduce the concept of final vertex.

Definition 5.1
A *final vertex* of a trajectory instance Tu_w^j w.r.t. a region *re* is a vertex in G that is traversed by the trajectory instance immediately before reaching *re*, denoted as $Tu_w^j.fv$ of *re*.

For example, in Fig. A.5b, v_3 is $Ref_1^1.fv$ w.r.t. re_4. Then, we introduce the format of a tuple for a reference Ref_i^j associated with a region *re*, if $(Ref_i^j.Rrs \cup Ref_i^j) \cap re \neq \emptyset$. Accordingly, there are two possible cases for Ref_i^j if it has a tuple corresponding to *re*: i) Ref_i^j passed *re* itself; ii) Ref_i^j did not pass *re* but $\exists Nref_{ik}^j \in Ref_i^j.Rrs$ s.t. $Nref_{ik}^j$ passed *re*.

For the first case, the tuple corresponding to *re* is stored as (*fv.id*, *fv.no*, *d.pos*, p_{total}, $p_{\max}$), where 1) *fv.id* (≥ 0) is the ID of $Ref_i^j.fv$ w.r.t. *re*; 2) *fv.no* indicates the position of *fv.id* in $E(Ref_i^j)$; 3) *d.pos* is the matching position of the $d.no^{th}$ relative distance in $\hat{D}(Ref_i^j)$, where $d.no{=}\gamma_{Ref_i^j}[fv.no]$; 4) with Ω defined as the subset of all trajectory instances $Ref_i^j \cup Ref_i^j.Rrs$ that overlap *re*, p_{total} is then the sum of the probabilities of all instances in Ω; and 5) $p_{\max} = Nref_{ik'}^j.p$

such that $\forall Nref_{ik}^{j} \in \Omega$: $Nref_{ik'}^{j}.p \geq Nref_{ik}^{j}.p$. If $\forall Nref_{ik}^{j} \in Ref_{i}^{j}.Rrs$, $Nref_{ik}^{j}$ does not overlap *re*, $p_{\max}$ is set to 0.

For the second case, each tuple has the form $(fv.id, p_{total}, p_{\max})$. Specifically, we set $fv.id = \infty$, which indicates that Ref_{i}^{j} itself did not traverse *re*, and p_{total} and $p_{\max}$ are the same as for the first case.

The tuple for a non-reference $Nref_{ik}^{j}$ w.r.t. *re* has the form (*rv.id*, *rv.no*, *ma.pos*), where 1) *rv.id* is the ID of the first vertex *rv* represented in $E_{ik}^{j}(Ma_h)$ (the h^{th} factor of $Com_E(Nref_{ik}^{j}, Ref_{i}^{j})$), such that $E_{ik}^{j}(Ma_h)$ contains $Nref_{ik}^{j}.fv$ of *re*; 2) *rv.no* indicates the position of *rv* in $E(Nref_{ik}^{j})$; and 3) *ma.pos* is the matching position of $E_{ik}^{j}(Ma_h)$ in $\hat{Com}_E(Nref_{ik}^{j}, Ref_{i}^{j})$. In the case when a factor "crosses" more than one region, we only keep the tuple for the region that the trajectory instance traverses first. In the case when *re* is the first region traversed by Tu_{w}^{j} or $Tu_{w}^{j}.fv$ of *re* is exactly $SV(Tu_{w}^{j})$, we store $(SV(Tu_{w}^{j}), 0, 0, p_{\max}, p_{total})$ if Tu_{w}^{j} is a reference, and we store $(SV(Tu_{w}^{j}), 0, 0)$ if Tu_{w}^{j} is a non-reference.

5.3 Probabilistic Queries

Based on StIU, three representative types of queries, namely probabilistic where, when, and range queries, can be performed.

Definition 5.2
Given a timestamp t, a probability α, and a compressed trajectory stream Tr^{n}, a probabilistic where query **where**(Tu^{j}, t, α) returns the set of mapped locations at time t of the instances $Tu_{w}^{j} \in Tu^{j}$ with $Tu_{w}^{j}.p \geq \alpha$. Each location is given as $\langle (v_s \rightarrow v_e), ndist \rangle$, where $(v_s \rightarrow v_e)$ is the edge traversed by Tu_{w}^{j}, and *ndist* is the network distance between v_s and the location at t.

Definition 5.3
(Probabilistic when query) Given a mapped location $\langle (v_s \rightarrow v_e), rd \rangle$, a probability α, and a compressed uncertain trajectory Tu^{j}, a probabilistic when query **when**$(Tu^{j}, \langle (v_s \rightarrow v_e), rd \rangle, \alpha)$ returns the set of timestamps, where *rd* is the relative distance of the location w.r.t. $(v_s \rightarrow v_e)$, and each timestamp t corresponds to a instance Tu_{w}^{j} of Tu^{j} with $Tu_{w}^{j}.p \geq \alpha$, such that Tu_{w}^{j} passed $\langle (v_s \rightarrow v_e), rd \rangle$ at t.

Example 5.1
(Probabilistic where and when query) Let the IDs of v_6 and v_7 be 228477 and 228478, respectively, and the length of the edge $(v_6 \rightarrow v_7)$ be 200. Given a query ***where***$(Tu^{1}, 5{:}21{:}25, 0.25)$ and assume that the time partition duration of StIU is 15 minutes, we locate the tuple $(5{:}15{:}26, 3, 23)$ through binary search, as its *t.start* is the closest timestamp to 5:21:25 with *t.start*$\leq$ 5:21:25, and then decompress $\hat{T}(Tu^{1})$ from its 23^{th} bit. In this way, we get

the result $\langle 228477 \rightarrow 228478, 150\rangle$ without full decompression. Similarly, given the road network partition shown in Fig. A.5b and a query ***when***$(Tu^1$, $\langle 228477 \rightarrow 228478, 0.75\rangle, 0.25)$, we search from v_5 according to the tuple $(228476, 3, 12, 1, 0.2)$ in StIU and return 5:21:25.

Definition 5.4
(Probabilistic range query) Given a query region RE, a timestamp t_q, and a collection of compressed uncertain trajectories ***Tu***, a probabilistic range query **range**$(\boldsymbol{Tu}, RE, t_q, \alpha)$ returns the set of uncertain trajectories $Tu^j (1 \leq j \leq M)$ in ***Tu***, such that $\sum_{Tu_w^j \in Tu^j \wedge Tu_w^j \cap RE \neq \emptyset} Tu_w^j.p \geq \alpha$ at t_q.

Example 5.2
(Probabilistic range query) A query ***range***$(\mathbf{Tu}, re_3 \cup re_4$, 5:05:25, $0.5)$ returns Tu^1, as Tu_1^1, Tu_2^1, Tu_3^1 overlaps $re_3 \cup re_4$ at 5:05:25 and $\sum_{w=1}^{3} Tu_w^1.p = 1 (> 0.5)$. However, a ***range***$(\mathbf{Tu}$, RE_1, 5:05:25, $0.5)$ returns empty due to $Tu^1 \cap RE_1 = \emptyset$.

5.4 Filtering and Validating Lemmas

When querying using the StIU, we can effectively avoid unnecessary decompression by exploiting p_{total} and $p_{\max}$ that are maintained for each reference.

Lemma 5.1
Given a query **when**$(Tu^j, \langle (v_s \rightarrow v_e), rd\rangle, \alpha)$, if $p_{\max} < \alpha$ holds for all the tuples of reference Ref_i^j in the StIU corresponding to the region where $\langle (v_s \rightarrow v_e), rd\rangle$ is located then we do not need to fully decompress Ref_i^j.

Proof. Let re be the region where $\langle (v_s \rightarrow v_e), rd\rangle$ is located, and Ω' be the subset of $Ref_i^j.Rrs$ that overlaps re. If $p_{\max} < \alpha$ holds for every tuple (under each time interval) associated with region re, it follows that $\forall Nref_{ik}^j \in \Omega'$, $Nref_{ik}^j.p < \alpha$ due to $Nref_{ik}^j.p \leq p_{\max}$. As a result, the timestamps of all non-references within Ω' will not be returned in accordance with the definition of a probabilistic when query. Therefore, Ref_i^j does not need to be fully decompressed. □

Example 5.3
(Filtering by Lemma 1) Given a query ***when***$(Tu^1, \langle (185191 \rightarrow 185192), 0.25\rangle, 0.5)$ in Fig. A.5b, Ref_1^1 does not need to be fully decompressed. This is because $Ref_1^1.p_{\max}$ w.r.t. re_3 where $\langle (185191 \rightarrow 185192), 0.25\rangle$ falls is 0.2, implying that $Nref_{1k}^1.p < 0.5\,(k = 1, 2)$.

Lemma 5.2
Given a spatial region RE, a timestamp t_q, and two edges $(v_s \rightarrow v_e)$ and $(v_{s'} \rightarrow v_{e'})$ where an uncertain trajectory instance Tu_i^j is located at timestamps

t_b and $t_{b'}$ $(t_b \leq t_q \leq t_{b'})$, (i) if the subpath sp from v_s to $v_{e'}$ satisfies $sp \in RE$ then Tu_i^j overlaps RE at t_q; (ii) if the subpath sp from v_s to $v_{e'}$ satisfies $sp \cap RE = \emptyset$ then Tu_i^j does not overlap RE at t_q.

Proof. Let the location where Tu_i^j is located at t_q be l. Then, l must be located on sp due to $t_b \leq t_q \leq t_{b'}$. Hence, Tu_i^j overlaps RE at t_q if $sp \in RE$; Tu_i^j does not overlap RE at t_q if $sp \cap RE = \emptyset$. □

Lemma 5.3
Given a query **range**($\boldsymbol{Tu}, RE, t_q, \alpha$) and a set Can^j that contains the instances of $Tu^j \in \boldsymbol{Tu}$ satisfying condition (i) in Lemma 5.2, if the sum of the probabilities of all the instances in Can^j is not smaller than α then Tu^j should be in the query result.

We omit the proof of Lemma 5.3 as it is straightforward.

Lemma 5.4
Given a query **range**($\boldsymbol{Tu}, RE, t_q, \alpha$), a region $re_{total}(RE \subseteq re_{total})$, and a set Can^j that contains all the instances of $Tu^j \in \boldsymbol{Tu}$ that overlap re_{total} during $[t_b, t_{b'}]$ $(t_b \leq t_q \leq t_{b'})$, if the sum of probabilities of all the instances in Can^j is smaller than α then Tu^j does not qualify as a query result.

Proof. Let Can'^j be a set of instances of Tu^j that overlaps RE at t_q. We have $Can'^j \subseteq Can^j$ due to $RE \subseteq re_{total}$ and $t_q \in [t_b, t_{b'}]$. As a result, the sum of probabilities of instances in Can'^j can not be greater than that in Can^j. Thus, if $\sum_{Tu_i^j \in Can^j} Tu_i^j.p < \alpha$ then Tu^j does not qualify as a query result. □

Example 5.4
(Filtering by Lemmas 5.2, 5.3, and 5.4) Given a query ***range***(**Tu**, $re_3 \cup re_4$, 5:05:25, 0.5) in Fig. A.5 and $\eta_p = \frac{1}{2048}$, we can get the subpath sp_1 from v_1 to v_4 after partially decompressing $T(Tu^1)$ and $E(Ref_1^1)$, where Ref_1^1 is located on $(v_1 \rightarrow v_2)$ at 5:03:25 and located on $(v_3 \rightarrow v_4)$ at 5:07:25. According to Lemma 5.2, we can ensure that Ref_1^1 must overlap $re_3 \cup re_4$ at 5:05:25 without decompressing $D(Ref_1^1)$. Since $Ref_1^1.p \geq 0.5$, Tu^1 can be directly returned by Lemma 5.3. Given a query ***range***(**Tu**, RE_1, 5:05:25, 0.5), we can infer that Ref_1^1, $Nref_{11}^1$ and $Nref_{12}^1$ do not overlap RE_1 without decompressing $D(Ref_1^1)$, $D(Nref_{11}^1)$ and $D(Nref_{12}^1)$ according to Lemma 5.2. This is because $sp_1 \cap RE_1 = \emptyset$ and $sp_2 \cap RE_1 = \emptyset$, where sp_2 is the subpath of $Nref_{11}^1$ from v_1 to v_{10}. Then, since the sum of the probabilities of instances in $Can^1 (= \emptyset)$ w.r.t. $re_{total} (= RE_1)$ is $0 (< 0.5)$, Tu^1 can be safely pruned by Lemma 5.4. Consider another example ***range***(**Tu**, RE_2,5:05:25, 0.8). Assume that only Ref_1^1 traversed re_1, Tu^1 can be safely pruned by Lemma 5.4 without checking any of its other instances, as the sum of the probabilities of instances in $Can^1 (= \{Ref_1^1\})$ w.r.t. $re_{total} (= re_1)$ is $0.75 (< 0.8)$.

Table A.5: Trajectory datasets

Datasets	Denmark	Chengdu	Hangzhou
Storage of NCUTs	0.97 GB	5.00 GB	20.20 GB
# of trajectories	266,913	1,956,640	1,807,895
# of trajectory instances	Average 9 (2 to 434)	Average 3 (2 to 192)	Average 13 (2 to 1,500)
# of edges per trajectory	Average 14 (2 to 139)	Average 11 (2 to 148)	Average 13 (2 to 189)
Default sample interval	1s	10s	20s

Table A.6: Road network information

Road network	# of edges	# of vertices	Out degree
Denmark	818,020	667,950	Average 2.449
Chengdu	125,929	88,868	Average 2.834
Hangzhou	85,949	61,581	Average 2.791

Due to the space limitation, the detailed algorithms for probabilistic where, when, and range quires are omitted.

6 Experiments

We report on extensive experiments aimed at evaluating the performance of the proposed framework.

6.1 Experimental Setting

Datasets. We use three real-life datasets, i.e., Denmark (DK), Chengdu (CD), and Hangzhou (HZ), as described in Table B.3, while the road network information is shown in Table A.6. The DK dataset is collected from 162 vehicles over about 2 years (Jan. 2007 to Dec. 2008) in Denmark. The CD dataset is collected from 14,864 taxis over one month (Aug. 2014) in Chengdu, China. The HZ dataset is collected from 24,515 taxis over one month (Nov. 2011) in Hangzhou, China.

Comparison Algorithm. As this is the first study on the compression of uncertain trajectories, we adapt the state-of-the-art work for the compression of accurate trajectories, i.e., the TED framework [40], to compress each uncertain trajectory instance while using the same to compress probability as our UTCQ. We omit bitmap compression [40], as it is time consuming and it is also applicable to UTCQ.

Parameter Setting. In the experiments, we study the effect on the performance of the parameters summarized in Table B.4. In addition, due to the use of the PDDP-tree [40], the error bound for representing the relative distance

Table A.7: Parameter ranges and default values

Parameter	Range
Number of instances	20%, 40%, 60%, 80%, **100%**
Trajectory length	20%, 40%, 60%, 80%, **100%**
Number of pivots	**1**, **2**, 3, 4, 5
Number of grid cells	8^2, 16^2, 32^2, $\mathbf{64^2}$, 128^2
Time partition duration (min)	10, 20, 30, 40, 50, **60**
Error bound of distance (meter)	$\frac{1}{8}$, $\frac{1}{16}$, $\frac{1}{32}$, $\frac{1}{64}$, $\mathbf{\frac{1}{128}}$
Error bound of probability	$\frac{1}{128}$, $\frac{1}{256}$, $\mathbf{\frac{1}{512}}$, $\frac{1}{1024}$, $\mathbf{\frac{1}{2048}}$

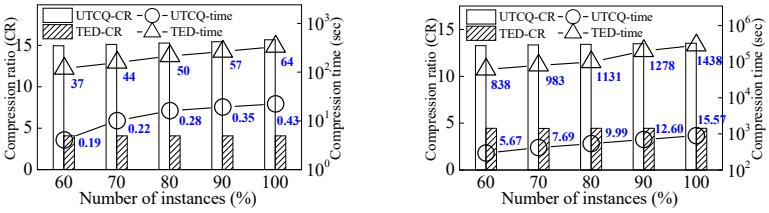
Table A.8: Comparison on three datasets

Datasets	UTCQ						
	Compression ratio						Time(s)
	Total	T	E	D	T′	p	
Denmark	14.342	7.685	14.861	26.171	15.843	7.111	23
Chengdu	11.867	3.128	13.589	15.141	18.061	7.111	135
Hangzhou	13.787	3.193	16.092	17.815	14.592	5.818	1031

Datasets	TED						
	Compression ratio						Time(s)
	Total	T	E	D	T′	p	
Denmark	4.439	4.545	11.888	9.143	1	7.111	1823
Chengdu	4.287	1.707	11.247	9.143	1	7.111	65310
Hangzhou	4.008	1.418	9.376	9.143	1	5.818	980447

η_D is set to $\frac{1}{128}$, while the error bound w.r.t. probability η_p is set to $\frac{1}{512}$ for the DK and CD datasets and to $\frac{1}{2048}$ for the HZ dataset. As the HZ dataset contains more instances for each uncertain trajectory, it is given a lower η_p.

Performance Metrics. For compression, we use the compression ratio, compression time, and maximum memory cost as the performance metrics. For query processing, we use the index size, query time, average difference, and F_1 score as performance metrics. All algorithms are implemented in C++ and run on a computer with Intel Core i9-9880H CPU (2.30 GHz) and 32 GB memory.

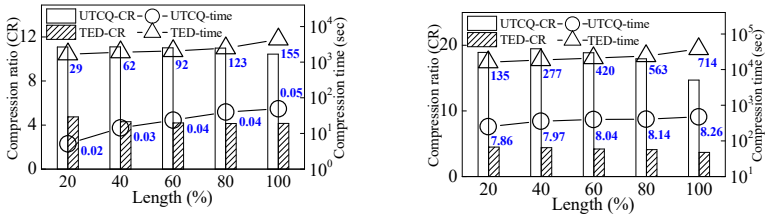


(a) DK dataset **(b)** HZ dataset

Figure A.6: Effect of the number of instances

6.2 Performance of Compression

We first compare UTCQ and TED in terms of compression ratio and time. Table A.8 shows the results, where T, E, D, T′, and p refer to the compression ratios of *time*, *edge*, *relative distance*, *time flag bit-string*, and *probability*, respectively, and Total denotes the total compression ratio. As observed, UTCQ outperforms TED more than 2–3 times in terms of compression ratio. The compression ratio of *time* offers evidence of the effectiveness of the SIAR scheme, while the compression ratios of *edge*, *relative distance*, and *time flag bit-string* reveal the effectiveness of referential compression. Moreover, the compression time of UTCQ is always more than 1–2 orders of magnitude smaller than that of TED, which validates the efficiency of UTCQ.

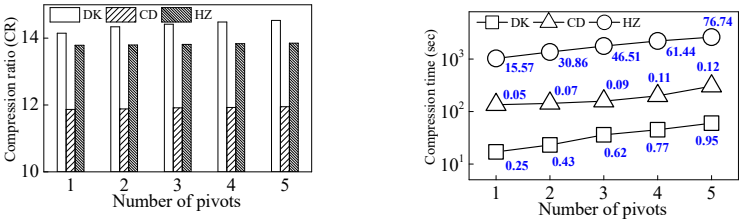


(a) DK dataset **(b)** HZ dataset

Figure A.7: Effect of the trajectory length

Effect of the Number of Instances. Fig. A.6 shows the compression ratio and time when varying the number of trajectory instances. Specifically, we filter the trajectories with fewer than 20 instances in the datasets. As observed, the compression ratio of UTCQ improves slightly when increasing the number of instances, while that of TED is unaffected. The reason is that the more instances we have, the more can be referentially represented by UTCQ. In contrast, TED compression is independent of the number of instances. Moreover, the time of UTCQ and TED grows with the number of instances, and UTCQ is 1–2 orders of magnitude faster than TED. Finally, the digits along with the compression time are the maximum memory cost during compression. As can be seen, the maximum memory cost of TED is always 1–2 orders of magnitude higher than that of UTCQ. This is because UTCQ processes uncertain trajectories one by one, while TED has to load all the $E(\cdot)$ for the preparation of matrix transformation and partitioning [40]. In addition, the memory cost grows with the number of instance, in accordance with the space complexity.

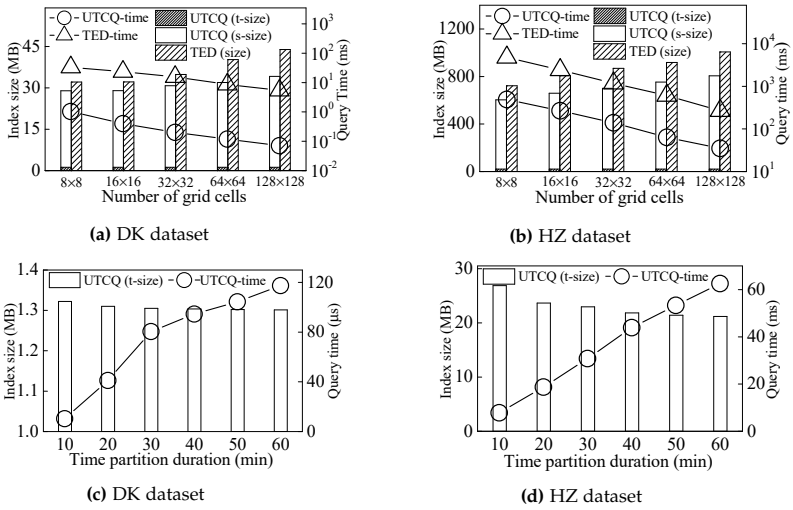
Effect of the Trajectory Length. Fig. A.7 reports the compression performance results for different lengths of trajectories. We eliminate trajectories with fewer than 20 edges and vary the trajectory length from 20% to 100% of the total number of edges. As can be seen, the compression ratios of UTCQ on both CD and HZ first increase slightly and then drop. This is because,



(a) Compression ratio **(b)** Compression time

Figure A.8: Effect of pivots

on the one hand, the compression ratio of *time* increases with the trajectory length, while, on the other hand, the referential compression performance drops due to the larger dissimilarity among longer sequences. Moreover, the compression ratio of TED decreases slightly as the highest bits of the entry path representation in one column [40] are more unlikely to be all 0. Finally, both the compression time and maximum memory cost increase slightly with the trajectory length, and UTCQ always uses 1–3 orders of magnitude less space and 1–2 orders of magnitude less time than TED.



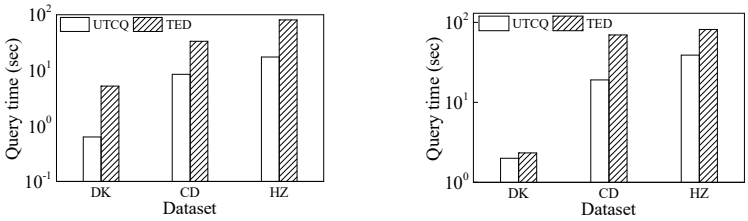
(a) DK dataset **(b)** HZ dataset

(c) DK dataset **(d)** HZ dataset

Figure A.9: Effect of spatial and temporal partition granularity on probabilistic range queries

Effect of the Number of Pivots. Fig. A.8 reports on the impact of the number of pivots on the compression performance. It can be seen that the compression ratio increases with the number of pivots used. The reason is that the more pivots, the higher the accuracy of the proposed similarity measure. On the other hand, the compression time and the maximal memory cost increase. As can be observed, the maximum memory cost on the CD

dataset is the smallest, since it has the shortest and the least instances for each uncertain trajectory on average among the three datasets. Specifically, we set the default pivot count to 1 on CD and HZ datasets, while set that to 2 on DK dataset. We do this because on DK dataset, the compression ratio improves significantly from 1 to 2 without reducing the efficiency considerably.



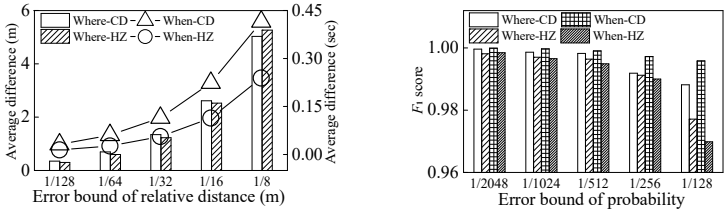
(a) Probabilistic where query

(b) Probabilistic when query

Figure A.10: Probabilistic where and when query performance

6.3 Query Performance

Probabilistic Range Query. Fig. A.9 reports the performance of probabilistic range queries when varying the spatio-temporal partition granularity. Here, we omit coverage of probabilistic where and when queries, as they are largely unaffected by variations in the spatio-temporal partitioning. Fig. A.9 indicates that the proposed index size is smaller than that of TED, which is due to the referential compression. It is clear that the query time decreases as road networks and time intervals are partitioned at finer granularities. Moreover, UTCQ is faster than TED, which is due to the index structure and the filtering and validating techniques.

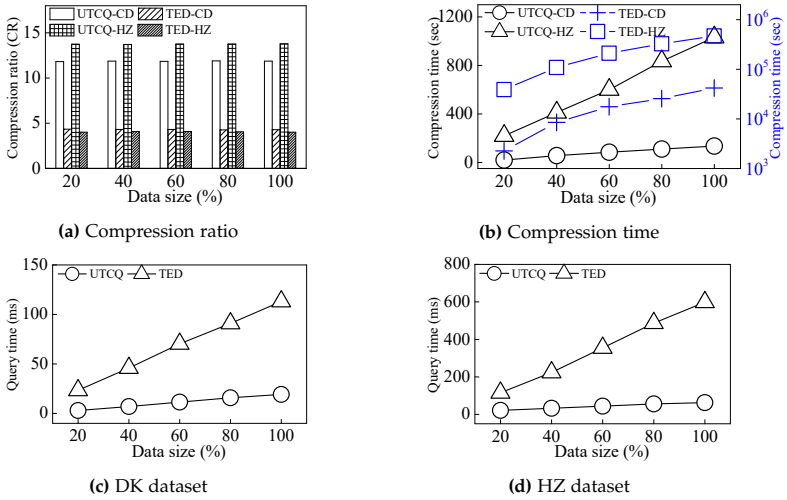


(a) Relative distance

(b) Probability

Figure A.11: Effect of error bound on query accuracy

Probabilistic Where and When Queries. We also report the performance (i.e., the query time) of probabilistic where and when queries. Fig. A.10 shows that UTCQ is faster than TED for both probabilistic where and when queries, due to the temporal index of StIU and Lemma 1 that are used for filtering. However, the superiority of UTCQ is not obvious on DK dataset for



(a) Compression ratio

(b) Compression time

(c) DK dataset

(d) HZ dataset

Figure A.12: Scalability of compression and query processing

probabilistic when query, because the query performance (i.e., the pruning ability of Lemma 1) relies on the distribution of the dataset.

Effect of Error Bound. Fig. A.11 studies the effect of the error bounds of the PDDP-tree [40] on the query accuracy, where the average difference and the $F_1 (= 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}})$ score are the performance metrics. Specifically, the average difference is the deviation between the query results derived from the original versus the compressed datasets. It is measured in meters (m) for probabilistic where queries and in seconds (sec) for probabilistic when queries. Fig. A.11 shows that the average difference is small, especially when η_D is set to the default value; and the F_1 score is always close to 1. These indicate that the error caused by PDDP-tree encoding is small. We omit the results on range queries because they achieve similar performance.

6.4 Scalability

Fig. A.12 reports on the scalability of compression and query processing, where the data size is varied from 20% to 100% of the total dataset storage size. Fig. A.12a shows that the compression ratios achieved by both UTCQ and TED are roughly independent of the dataset size. This is because the compression ratio is unaffected by the number of uncertain trajectories, but rather depends on the number and the lengths of instances. In Fig. A.12b, the compression time of UTCQ is measured by the left y axis in black color, while that of TED is measured by the right y axis in blue color. Two axes are used to show more clearly the time for both solutions that differ substantially. We see that the compression time of UTCQ increases linearly as it processes trajectories one

by one, while that of TED increases super linearly due to its matrix operations. As expected, the query time of both UTCQ and TED increase linearly with the growth in the data size, as shown in Figs. A.12c and A.12d.

7 Related Work

Trajectory compression can be classified into raw data-oriented compression and road network-embedded compression.

7.1 Raw Data-oriented Compression

Raw data-oriented compression techniques are designed to compact trajectories that have not been map-matched. A typical approach is to use trajectory simplification that approximates an original trajectory by a subsequence of the trajectory while attempting to minimize the information loss according to certain distance measures. The *Bellaman* [1] method uses dynamic programming to find a subsequence with the minimum spatial distance error. The *MRPA* algorithm [8] employs a distance measure called Integral Square Synchronous Euclidean Distance to simplify trajectories.To minimize the simplification error under a storage budget, Min-Error [25] is proposed. It protects the direction information of trajectories using a direction-based error measurement to detect the sharp change of directions of trajectories. In addition, trajectory simplication methods can be classified into offline methods [1, 8, 25] and online methods [4, 22–24, 29], where the offline methods require that full trajectories are available before compression starts, while the online methods can compress trajectories in streaming settings. Comprehensive experimental evaluations of trajectory simplification techniques are available [41].

Approaches based on other strategies also exist. Philippe et al. [9] propose two strategies, Single Trajectory Delta compression and Cluster-based compression. The former compresses each single trajectory by encoding the deviation between successive values. The latter clusters similar subtrajectories and only stores one summary trajectory per cluster. Wandelt and Sun [36] propose a lossless compression technique for 4D trajectories that exploits the similarities between subtrajectories by predicting the next point in a trajectory based on previous trajectories. Cai et al. [3] assume that moving objects are likely to maintain a certain mode during a period and extract this mode as a state vector based on sampling. Zhao et al. [44] construct a reference trajectory set and represent a raw trajectory as a concatenation of a series reference trajectories within a given spatio-temporal deviation threshold. More recently, Gao et al. [11] also study semantic-based compression.

The above methods do not consider the road network embedding and are not competitive in our setting.

7.2 Road Network-embedded Compression

Road network-embedded compression leverages an underlying road network to achieve better trajectory compression. GPS points are first map-matched to road segments [2, 14, 15, 26]. Since a sequence of successive points can often be mapped to, and represented by, the same segment, spatial redundancy can be reduced, which yields a higher compression ratio. Auxiliary information, such as frequent travel paths and shortest travel paths are also utilized in existing studies [13, 16, 18, 31] to improve compression. Road network-embedded compression can be classified as spatial compression or spatio-temporal compression.

Spatial compression. Krogh et al. [18] compress a trajectory by only storing the first and last edge of each shortest path in the trajectory. Koide et al. [17] present a compression technique for spatial information of trajectories and support the retrieval of subpaths. Specifically, they store path information in a Huffman-based Wavelet Tree (HWT) that counts the frequency of each label in advance. Chen et al. [5] compress trajectories by retaining out-edges with remarkable heading changes. Sui et al. [32] assign each GPS point to the middle point of a segment and propose a road-network partitioning strategy on which the compression ratio depends.

Spatio-temporal compression. Most existing studies [6, 12, 13, 16, 31] represent the temporal information of trajectories as pairs (d, t), where d is the network distance traveled at the timestamp t since the start of the trajectory. Sun et al. [13, 31] propose a two-stage spatial compression algorithms encompassing shortest path and frequent sub-trajectory compression. Ji et al. [16] encode outgoing road segments clockwise based on a pre-computed clockwise code table. Chen et al. [6] focus mainly on reducing the frequency of data transmission and adopt existing integer encoding approaches [19, 39] to compress trajectories.

The work closest to ours is TED [40]. Instead of representing (d, t) together, TED represents them individually, leading to a lossless compression in terms of t and a higher compression ratio. In addition, it provides an index structure for facilitating queries of compressed trajectories. However, TED cannot solve our problem efficiently as it is not designed to take the uncertainty of trajectories into account. In contrast, we exploit the similarities between uncertain trajectory instances to achieve high performance. To the best of our knowledge, we propose the first framework for compressing and querying uncertain trajectories. Moreover, we propose an effective index structure to support efficient queries against compressed uncertain trajectories.

8 Conclusion

We propose a novel framework for compressing and querying uncertain trajectories. We referentially represent uncertain trajectories by exploiting similarities between trajectory instances. To achieve this, we propose a reference selection algorithm that uses a new similarity measure, as well as several referential representation formats that make it possible to represent trajectories at a high compression ratio. As part of this, we propose an effective representation of the temporal information in trajectories that addresses fluctuations in the sampling time intervals as seen in real-life data. In addition, we propose an effective index for compressed trajectories and develop filtering techniques to accelerate probabilistic where, when, and range queries over compressed data, where *flag array* and *original array* structures are constructed to extract necessary information without full decompression. Extensive experiments conducted on three real datasets show that the UTCQ framework is 2–3 times better than the state-of-the-art method in terms of compression ratio, uses 1–3 orders of magnitude less memory, is 1–2 orders of magnitude faster in terms of compression time, and is always faster in terms of query time. In the future, it is of interest to introduce a multiple-order representation that may further improve the compression performance, and it may also be possible to develop techniques that can recover a non-reference without decompressing its reference.

References

[1] R. Bellman and B. Kotkin, "On the approximation of curves by line segments using dynamic programming. ii," RAND CORP SANTA MONICA CALIF, Tech. Rep., 1962.

[2] M. Bierlaire, J. Chen, and J. Newman, "A probabilistic map matching method for smartphone GPS data," *TRANSPORT RES C-EMER*, vol. 26, pp. 78–98, 2013.

[3] Z. Cai, F. Ren, J. Chen, and Z. Ding, "Vector-based trajectory storage and query for intelligent transport system," *TITS*, vol. 19, no. 5, pp. 1508–1519, 2017.

[4] W. Cao and Y. Li, "Dots: An online and near-optimal trajectory simplification algorithm," *J Syst Softw*, vol. 126, pp. 34–44, 2017.

[5] C. Chen, Y. Ding, X. Xie, S. Zhang, Z. Wang, and L. Feng, "Trajcompressor: An online map-matching-based trajectory compression framework leveraging vehicle heading direction and change," *IEEE trans Intell Transp Syst*, vol. 21, no. 5, pp. 2012–2028, 2019.

[6] J. Chen, Z. Xiao, D. Wang, D. Chen, V. Havyarimana, J. Bai, and H. Chen, "Toward opportunistic compression and transmission for private car trajectory data collection," *IEEE Sens. J.*, vol. 19, no. 5, pp. 1925–1935, 2018.

[7] L. Chen, Y. Gao, X. Li, C. S. Jensen, and G. Chen, "Efficient metric indexing for similarity search," in *ICDE*, 2015, pp. 591–602.

[8] M. Chen, M. Xu, and P. Franti, "A fast $o(n)$ multiresolution polygonal approximation algorithm for GPS trajectory simplification," *TIP*, vol. 21, no. 5, pp. 2770–2785, 2012.

[9] P. Cudre-Mauroux, E. Wu, and S. Madden, "Trajstore: An adaptive storage system for very large trajectory data sets," in *ICDE*, 2010, pp. 109–120.

[10] S. Deorowicz and S. Grabowski, "Robust relative compression of genomes with random access," *Bioinformatics*, vol. 27, no. 21, pp. 2979–2986, 2011.

[11] C. Gao, Y. Zhao, R. Wu, Q. Yang, and J. Shao, "Semantic trajectory compression via multi-resolution synchronization-based clustering," *Knowl Based Syst*, vol. 174, pp. 177–193, 2019.

[12] Y. Gao, B. Zheng, G. Chen, Q. Li, C. Chen, and G. Chen, "Efficient mutual nearest neighbor query processing for moving object trajectories," *Inf. Sci*, vol. 180, no. 11, pp. 2176–2195, 2010.

[13] Y. Han, W. Sun, and B. Zheng, "Compress: A comprehensive framework of trajectory compression in road networks," *TODS*, vol. 42, no. 2, p. 11, 2017.

[14] G. Hu, J. Shao, F. Liu, Y. Wang, and H. Shen, "If-matching: Towards accurate map-matching with information fusion," *TKDE*, vol. 29, no. 1, pp. 114–127, 2016.

[15] G. R. Jagadeesh and T. Srikanthan, "Probabilistic map matching of sparse and noisy smartphone location data," in *ITSC*, 2015, pp. 812–817.

[16] Y. Ji, Y. Zang, W. Luo, X. Zhou, Y. Ding, and L. M. Ni, "Clockwise compression for trajectory data under road network constraints," in *ICBDA*, 2016, pp. 472–481.

[17] S. Koide, Y. Tadokoro, C. Xiao, and Y. Ishikawa, "CiNCT: Compression and retrieval for massive vehicular trajectories via relative movement labeling," in *ICDE*, 2018, pp. 1097–1108.

[18] B. Krogh, C. S. Jensen, and K. Torp, "Efficient in-memory indexing of network-constrained trajectories," in *SIGSPATIAL*, 2016, pp. 17–26.

[19] D. Lemire and L. Boytsov, "Decoding billions of integers per second through vectorization," *SOFTWARE PRACT EXPER*, vol. 45, no. 1, pp. 1–29, 2015.

[20] Y. Li, K. Gai, L. Qiu, M. Qiu, and H. Zhao, "Intelligent cryptography approach for secure distributed big data storage in cloud computing," *Inf. Sci*, vol. 387, pp. 103–115, 2017.

[21] Y. Li, H. Zhang, X. Liang, and B. Huang, "Event-triggered-based distributed cooperative energy management for multienergy systems," *IEEE T IND INFORM*, vol. 15, no. 4, pp. 2008–2022, 2018.

[22] X. Lin, J. Jiang, S. Ma, Y. Zuo, and C. Hu, "One-pass trajectory simplification using the synchronous Euclidean distance," *arXiv preprint arXiv:1801.05360*, 2018.

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

[24] J. Liu, K. Zhao, P. Sommer, S. Shang, B. Kusy, J.-G. Lee, and R. Jurdak, "A novel framework for online amnesic trajectory compression in resource-constrained environments," *TKDE*, vol. 28, no. 11, pp. 2827–2841, 2016.

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

[26] Y. Lou, C. Zhang, Y. Zheng, X. Xie, W. Wang, and Y. Huang, "Map-matching for low-sampling-rate gps trajectories," in *SIGSPATIAL*, 2009, pp. 352–361.

[27] J. Muckell, P. W. Olsen, J.-H. Hwang, C. T. Lawson, and S. Ravi, "Compression of trajectory data: a comprehensive evaluation and new approach," *GeoInf*, vol. 18, no. 3, pp. 435–460, 2014.

[28] D. A. Peixoto, H. Q. V. Nguyen, B. Zheng, and X. Zhou, "A framework for parallel map-matching at scale using Spark," *DISTRIB PARALLEL DAT*, vol. 37, no. 4, pp. 697–720, 2019.

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

[30] W. Sebastian and L. Ulf, "Fresco: Referential compression of highly similar sequences," *TCBB*, vol. 10, no. 5, pp. 1275–1288, 2013.

[31] R. Song, W. Sun, B. Zheng, and Y. Zheng, "Press: A novel framework of trajectory compression in road networks," *PVLDB*, vol. 7, no. 9, pp. 661–672, 2014.

[32] P. Sui and X. Yang, "A privacy-preserving compression storage method for large trajectory data in road network," *J. Grid Comput.*, vol. 16, no. 2, pp. 229–245, 2018.

[33] J. Teuhola, "A compression method for clustered bit-vectors," *INFORM PROCESS LETT*, vol. 7, no. 6, pp. 308–311, 1978.

[34] S. J. van Schaik and O. de Moor, "A memory efficient reachability data structure through bit vector compression," in *SIGMOD*, 2011, pp. 913–924.

[35] S. Wandelt and U. Leser, "Adaptive efficient compression of genomes," *ALGORITHM MOL BIOL*, vol. 7, no. 1, p. 30, 2012.

[36] S. Wandelt and X. Sun, "Efficient compression of 4D-trajectory data in air traffic management," *TITS*, vol. 16, no. 2, pp. 844–853, 2014.

[37] J. Wang, J. Feng, and G. Li, "Trie-join: Efficient trie-based string similarity joins with edit-distance constraints," *PVLDB*, vol. 3, no. 1–2, pp. 1219–1230, 2010.

[38] L.-Y. Wei, Y. Zheng, and W.-C. Peng, "Constructing popular routes from uncertain trajectories," in *KDD*, 2012, pp. 195–203.

[39] H. Yan, S. Ding, and T. Suel, "Inverted index compression and query processing with optimized document ordering," in *WWW*, 2009, pp. 401–410.

[40] X. Yang, B. Wang, K. Yang, C. Liu, and B. Zheng, "A novel representation and compression for queries on trajectories in road networks," *TKDE*, vol. 30, no. 4, pp. 613–629, 2017.

[41] D. Zhang, M. Ding, D. Yang, Y. Liu, J. Fan, and H. Shen, "Trajectory simplification: an experimental study and quality analysis," *PVLDB*, vol. 11, no. 9, pp. 934–946, 2018.

[42] H. Zhang, Y. Li, D. W. Gao, and J. Zhou, "Distributed optimal energy management for energy internet," *IEEE T IND INFORM*, vol. 13, no. 6, pp. 3081–3097, 2017.

[43] Z. Zhang, M. Hadjieleftheriou, B. C. Ooi, and D. Srivastava, "Bed-tree: an all-purpose index structure for string similarity search based on edit distance," in *SIGMOD*, 2010, pp. 915–926.

[44] 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 *KDD*, 2018, pp. 2797–2806.

[45] Y. Zheng, "Trajectory data mining: an overview," *TIST*, vol. 6, no. 3, pp. 1–41, 2015.

Paper B

TRACE: Real-time Compression of Streaming Trajectories in Road Networks

Tianyi Li, Lu Chen, Christian S. Jensen, Torben Bach Pedersen

The paper has been published in the
International Conference on Very Large Data Bases (PVLDB), pp. 1175–1187, 2021.

The layout has been revised.

Abstract

The deployment of vehicle location services generates increasingly massive vehicle trajectory data, which incurs high storage and transmission costs. A range of studies target offline compression to reduce the storage cost. However, to enable online services such as real-time traffic monitoring, it is attractive to also reduce transmission costs by being able to compress streaming trajectories in real-time. Hence, we propose a framework called TRACE that enables compression, transmission, and querying of network-constrained streaming trajectories in a fully online fashion. We propose a compact two-stage representation of streaming trajectories: a speed-based representation removes redundant information, and a multiple-references based referential representation exploits subtrajectory similarities. In addition, the online referential representation is extended with reference selection, deletion and rewriting functions that further improve the compression performance. An efficient data transmission scheme is provided for achieving low transmission overhead. Finally, indexing and filtering techniques support efficient real-time range queries over compressed trajectories are developed. Extensive experiments with real-life and synthetic datasets evaluate the different parts of TRACE, offering evidence that it is able to outperform the existing representative methods in terms of both compression ratio and transmission cost.

1 Introduction

Massive volumes of vehicle trajectories are being accumulated at an unprecedented scale with the proliferation of GPS-enabled devices and mobile internet connectivity. This yields high storage and transmission costs for trajectories. Hence, trajectory compression that addresses these aspects has attracted attention [3–5, 8, 11, 13, 16, 17, 19, 27, 32–34, 41, 44, 45]. However, most existing studies target offline compression [11, 13, 16, 17, 19, 33, 34, 41, 44, 45]. They generally compress an entire trajectory after all the GPS points are collected, which may not be realistic for resource-constrained GPS-enabled devices. In particular, offline compression incurs high communication overheads or data loss because the data needs to be transmitted to the location where the compression is performed. In contrast, with online compression, GPS points are compressed as they arrive in real-time, thus enabling a broader range of applications, while saving both storage and transmission costs [3, 4, 22, 27].

To enable compression on diverse devices with variable computing capabilities, we employ vehicular edge computing (VEC) to compress trajectories in real-time [3]. A VEC architecture has three layers: a smart vehicle layer, an edge server layer, and a centralized cloud layer [29], as shown in Figure B.1. The smart vehicle layer delivers raw GPS data to the edge server layer that encompasses software-defined networking (SDN) based roadside units (SRSUs), which possess the computational and storage capabilities needed for trajectory

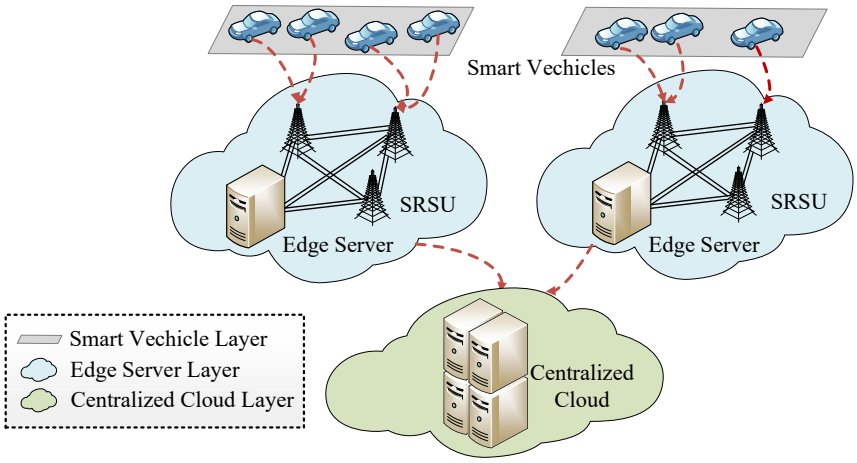


Figure B.1: Vehicular edge computing architecture.

compression. The centralized cloud layer collects and stores the compressed trajectories from the edge server layer to provide multiple services.

Although several studies [3–5, 32] consider online network-constrained trajectory compression, two challenges remain to be tackled. *The first challenge is how to obtain a concise and accurate representation of trajectories.* Existing studies obtain a compact representations in part by discarding information [3–5, 32], e.g., the exact locations of trajectories. This renders the resulting trajectories inaccurate and reduces their usability. *The second challenge is how to compress trajectories in real-time with low transmission costs.* Some previous proposals for streaming trajectory compression rely on offline training of prediction models using historical data, enabling them to omit data that can be predicted within a certain error bound [5, 32]. However, movement patterns on even the same road vary across time [12, 24], necessitating frequent re-training and incurring high transmission cost for delivering re-trained models.

To address the above two challenges, we propose a new framework for online TRAjectory ComprEssion (TRACE). The goals of TRACE are to achieve high compression ratios and low transmission costs with acceptable time delays. To realize these, we first present a speed-based trajectory representation on the basis of UTCQ [19]. This representation removes redundant information while enabling decompression of trajectories by capturing growth rates of accumulative distances and vehicle speeds. Further, as existing studies indicate that subtrajectories from different streaming trajectories are likely to exhibit co-movement patterns during the same time periods [12], we make it possible to exploit the similarity between subtrajectories from different streaming trajectories by means of so-called referential compression. Next, we develop an effective online referential representation and a reference selection technique based on so-called k-mer matching, which employs hashing to

identify matching subsequences [25]. To keep memory consumption low, we design a reference deletion algorithm that removes references that have not been used for some time. To be able to adapt to variable movement patterns, we present a reference rewriting algorithm that updates the references in real-time. Further, we provide a real-time data transmission scheme that targets low-overhead transmission of trajectory data. Finally, we develop an index structure and filtering techniques that facilitate real-time range querying of compressed trajectories.

In summary, our main contributions are as follows:

- We propose a new real-time streaming vehicle trajectory compression, transmission, and querying framework. To the best of our knowledge, this is the first such framework that does not depend on offline training and discard any data.
- We develop a concise speed-based representation and a k-mer matching based referential representation that use multiple references to capture the similarities between subtrajectories. A reference selection technique and reference deletion and rewriting functions are provided that further improve compression performance.
- We provide an effective data transmission scheme that reduces transmission overhead and supports decoding at the centralized cloud. We also propose an index structure and filtering techniques to accelerate real-time query processing.
- Extensive experiments offer insight into the workings of the different parts of the framework and show that it is able to outperform three baselines in terms of compression ratio and transmission cost.

The rest of the paper is organized as follows. We present preliminaries in Section 2 and give an overview of the proposed framework in Section 3. Section 4 details the representation. Section 5 presents the encoding and transmission schemes, and Section 6 covers the index structure and query processing. Section 7 reports the experimental results. Section 7 reviews related work, and Section 9 concludes and offers directions for future work.

2 Preliminaries

We proceed to introduce preliminary definitions and algorithms. Table B.1 summarizes frequently used notation.

2.1 Data Model

A **raw trajectory** is a series of **raw GPS points** $p = ((x, y), t)$, where x is longitude, y is latitude, and t is a timestamp. $Tp^1 = \langle p_0, \cdots, p_7 \rangle$ in Figure B.2a

Table B.1: Frequently used notation.

Notation	Description
Tr	a set of streaming trajectories
Tr^n	a streaming trajectory in **Tr**
l_i	the i^{th} mapped GPS point
$sp(Tr^n)$	the path traversed by Tr^n
$ad(Tr^n)$	the accumulative distance sequence of Tr^n
$t(Tr^n)$	the time sequence of Tr^n
$SV(Tr^n)$	the start vertex of Tr^n
$E(Tr^n)$	the outgoing edge number sequence of Tr^n
$RD(Tr^n)$	the first relative distance of Tr^n
$GD(Tr^n)$	the growth rates of accumulative distances of Tr^n
$V(Tr^n)$	the speed sequence of Tr^n
$E(Tr^n)[i]$	the i^{th} outgoing edge number of $E(Tr^n)$
$E(Tr_i^n)$	the outgoing edge numbers arriving at $t(Tr^n)[i]$
Ref	a reference streaming trajectory
Nref	a non-reference streaming trajectory
$Com_\phi(Nref)$	the referential representation of *Nref*
G_o	the reference set at timestamp t_o
$G_o[i].f$	the freshness of the i^{th} reference in G_o
$G_o[i].tl$	the latest visiting timestamp of the i^{th} reference in G_o
F_o	the sum of freshness of references in G_o
FA	a factor matrix
$\hat{seq}$	the binary code of a sequence *seq*

is an example of a raw trajectory. A **road network** is modeled as a directed spatial graph $G = (V, E)$, where V is a set of geo-located vertices v denoting intersections or end points, and E is a set of directed edges $e = (v_i \rightarrow v_j)$. Figure B.2 gives a road network example. A **mapped GPS point** l is a network-constrained point in a road network G obtained by map-matching [35]. It is represented as $((v_i \rightarrow v_j), nd(v_i, l), t)$, where $nd(v_i, l)$ is the network distance between v_i and l on the edge $(v_i \rightarrow v_j)$ and t is a timestamp. In Figure B.2a, $l_0 = ((v_0 \rightarrow v_1), 50, 7{:}03{:}25)$ is a mapped GPS point. We also denote a mapped GPS point as $((v_i \rightarrow v_j), nd(v_i, l))$ when the timestamp t is not considered.

Definition 2.1
Given two vertices v_s and v_e in a road network G, a **path** sp is a sequence of connected edges $(v_i \rightarrow v_j)$ that starts from v_s and ends at v_e, i.e., $sp = \langle (v_s \rightarrow v_0), \cdots, (v_{n-1} \rightarrow v_e) \rangle$.

Definition 2.2
A **streaming network-constrained trajectory** Tr^n is modeled as an infinite, time-ordered sequence of mapped GPS points L^n with an infinite path $sp(Tr^n)$ traversed by Tr^n.

Figure B.2 gives an example of a set $\mathbf{Tr} = \{Tr^1, Tr^2, Tr^3\}$ of three streaming network-constrained trajectories, where, e.g., $sp(Tr^1) = \langle (v_0 \rightarrow v_1), \cdots, (v_{11} \rightarrow$

$v_{12}), \cdots\rangle$ and $L^1 = \{l_0, l_1, \cdots, l_7, \cdots\}$. **Tr** uses two road-network distances as defined next.

Definition 2.3
The **accumulative distance** of a streaming trajectory Tr^n at its i^{th} timestamp $t(Tr^n)[i]$, denoted as $ad(Tr^n)[i]$, is the network distance $nd(v_s, l_i)$ along the path $(v_s \rightarrow v_e), \cdots, (v_{s^*} \rightarrow v_{e^*})$, where l_i is located on $(v_{s^*} \rightarrow v_{e^*})$ and $(v_s \rightarrow v_e)$ is the first edge traversed by Tr^n. The accumulative distance sequence $ad(Tr^n)$ of a streaming trajectory Tr^n contains the trajectory's accumulative distance at each timestamp.

Definition 2.4
Given a mapped GPS point $((v_s \rightarrow v_e), nd(v_s, l))$, the **relative distance** rd of l w.r.t. $(v_s \rightarrow v_e)$ is the ratio of $nd(v_s, l)$ to the length of $(v_s \rightarrow v_e)$ (denoted as $|(v_s \rightarrow v_e)|$).

In Figure B.2a, given $nd(v_0, l_1) = 150$, we have $ad(Tr^1)[1] = 150$. Given $|(v_0 \rightarrow v_1)| = 100$, rd of l_0 w.r.t. $(v_0 \rightarrow v_1)$ is 0.5. In the rest of the paper, we simply use "trajectory" instead of "network-constrained trajectory" when this does not cause ambiguity.

2.2 UTCQ Representation

Trajectory representation transforms network-constrained trajectories into a format with small entropy to achieve a high compression ratio [11, 13, 16, 19, 33, 41]. The UTCQ representation [19] is designed for compressing uncertain trajectories. Due to the uncertainty, a raw trajectory can be transformed to multiple trajectory instances by probabilistic map-matching [2]. UTCQ first adapts the representative trajectory representation TED [41] to express a trajectory instance Tr^n as a start vertex $SV(Tr^n)$, an edge sequence $E(Tr)$, a relative distance sequence $D(Tr)$, a time flag bit-string $T'(Tr)$, and a timestamp sequence $T(Tr)$.

Example 2.1
Assuming that the default sample interval of **Tr** in Figure B.2 is 20s, the UTCQ representation of Tr^1 is i) $SV(Tr^1) = 44183$; ii) $E(Tr^1) = \langle 2, 2, 1, 1, 4, 1, 2, 1, 2, 3, 2, 3, 0\rangle$; iii) $D(Tr^1) = \langle 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.75\rangle$; iv) $T'(Tr^1) = \langle 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 1\rangle$ and v) $T(Tr^1) = \langle 7{:}03{:}25, 0, 1, 0, 0, -1, 5, 0\rangle$. The single 0 and the last 3 (3 is the outgoing edge number of $(v_{11} \rightarrow v_{12})$ w.r.t. v_{11}) in $E(Tr^1)$ indicate that $(v_{11} \rightarrow v_{12})$ has two mapped GPS points, l_6 and l_7. $T'(Tr)$ is introduced to map relative distances in $D(Tr)$ to outgoing edge numbers in $E(Tr)$ for decompression [41].

Next, UTCQ exploits the similarity between trajectory instances of a single uncertain trajectory and provides **referential representations** for the edge sequence $E(Tr)$, the distance sequence $D(Tr)$, and the time flag bit-string

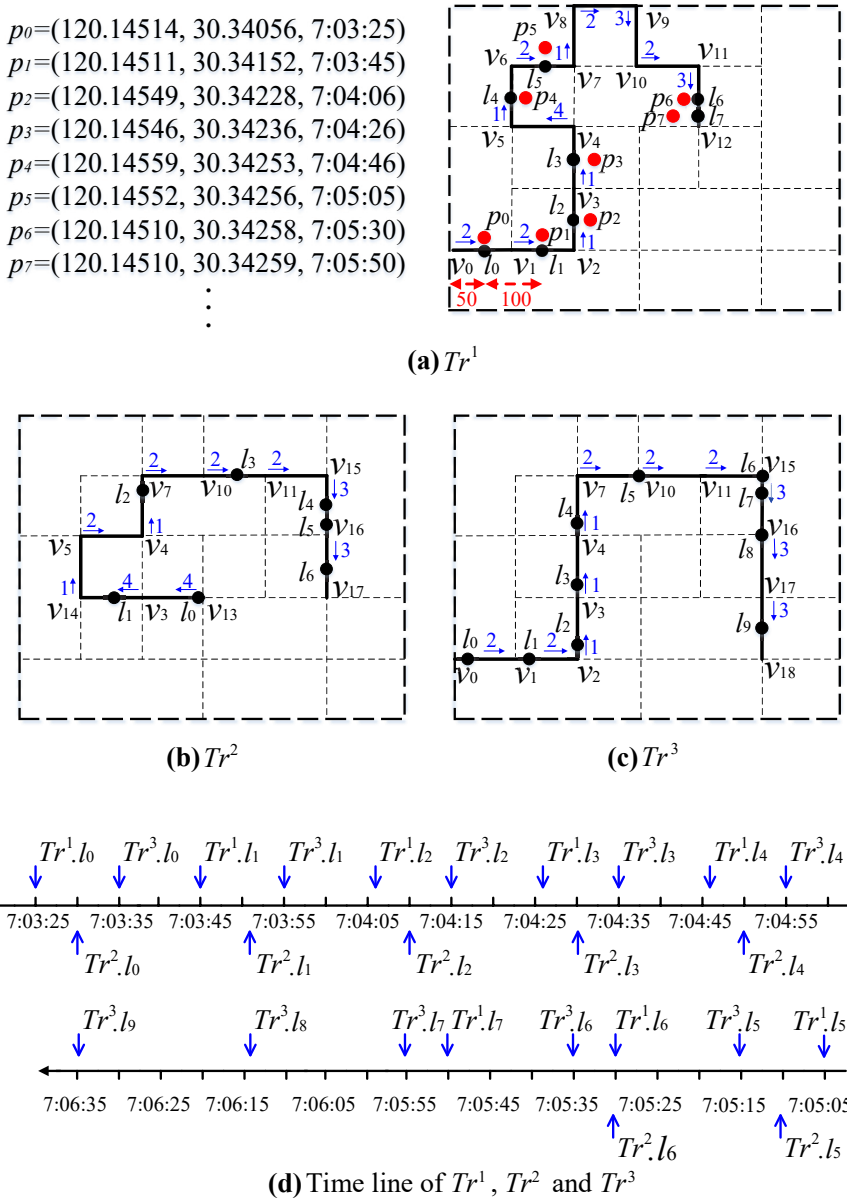


Figure B.2: A streaming network-constrained trajectory set $\mathbf{Tr} = \{Tr^1, Tr^2 \text{ and } Tr^3\}$.

$T'(Tr)$. It encodes the differences of an input sequence w.r.t. a reference sequence. (i.e., the more similar the sequences are, the higher the compression ratio) [9, 31, 39].

Definition 2.5
Given an input sequence $\phi(Nref)$ (also called a non-reference) and its corresponding reference $\phi(Ref)$, $\phi(Nref)$ can be represented as a list of W **factors**, i.e., $Com_\phi(Nref) = \langle \phi(Ma_w) | 0 \leq w < W \rangle$, where a factor $\phi(Ma_w)$ denotes a subsequence in $\phi(Nref)$.

Here, ϕ identifies the to-be-represented sequence of a trajectory. For example, $Com_E(Nref)$ is the referential representation of the edge sequence $E(Nref)$. UTCQ adopts the (S, L, M) format to encode each factor in $Com_\phi(Nref)$. Specifically, S is the *start position* of the subsequence in the reference, L is the *length* of the subsequence, and M is the first *mis-matched character* following the subsequence. For example, in Figure B.2, we have $E(Tr^2) = \langle 4, 4, 1, 2, 1, 2, 2, 2, 3, 0, 3 \rangle$ and $E(Tr^3) = \langle 2, 2, 1, 1, 1, 2, 2, 2, 3, 0, 3, 3 \rangle$. Let Tr^3 be a non-reference *Nref* and Tr^2 be its corresponding reference *Ref*. We get $Com_E(Nref) = \langle (5, 2, 1), (2, 1, 1), (5, 6, 3) \rangle$. Note that UTCQ only assigns one reference to each non-reference.

2.3 *k*-mer Matching

Developed for genome sequences, k-mer matching is an effective strategy for high-speed referential compression [25].

Definition 2.6
A ***k*-mer** ϕ_i^n is a subsequence of fixed length k of $\phi(Tr^n)$, i.e., $\phi_i^n = \{\phi(Tr^n)[i], \phi(Tr^n)[i+1], ..., \phi(Tr^n[i+k-1])\}$.

Given a reference $\phi(Ref)$ and a non-reference $\phi(Nref)$, k-mer matching employs a hash table H, to efficiently obtain the referential representation $Com_\phi(Nref)$. Each factor of $Com_\phi(Nref)$ is of the form (S, L, M), where L is the length of the matched subsequence. We highlight that k is different from L; thus, k is kept fixed during compression. To be specific, for a reference $\phi(Ref)$, k-mer matching first computes the hash key of each k-mer in $\phi(Ref)$ by a given hash function. Next, each k-mer is stored with its start position S in $\phi(Ref)$ in H. For a non-reference $\phi(Nref)$, k-mer matching greedily finds the longest prefix of $\phi(Nref)$ that exists in $\phi(Ref)$ with the help of H. In particular, it calculates the hash key of each k-mer in $\phi(Nref)$ and checks whether a matched subsequence exists in H. If it exists, it continues to match the subsequent characters in $\phi(Nref)$ and $\phi(Ref)$ until an un-matched character M occurs. The procedure implies that L can be an **arbitrary value** with $L \geq k$, meaning that k can remain fixed during compression. It also means that the correctness of k-mer matching is un-affected by k.

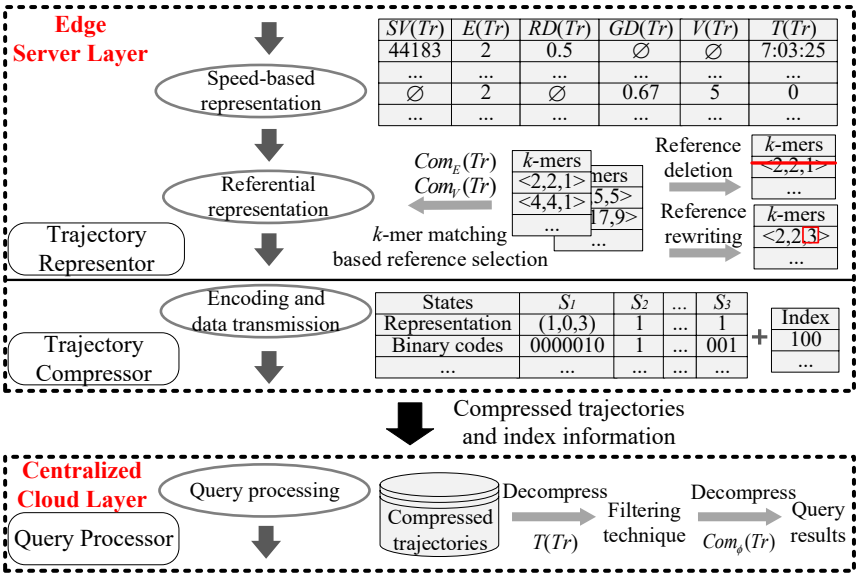


Figure B.3: TRACE framework.

Example 2.2
Consider $E(Tr^1)$ as a reference and $E(Tr^3)$ as a non-reference. Assuming that the current timestamp is 7:04:55, we have four k-mers for $E(Tr^1)$, i.e., $E_0^1 = (2,2,1)$, $E_1^1 = (2,1,1)$, $E_2^1 = (1,1,4)$, and $E_3^1 = (1,4,1)$, each of which is mapped to a hash table with its corresponding start position in $E(Tr^1)$. To find the longest prefix of $E(Tr^3)$ in $E(Tr^1)$, we get the first k-mer of $E(Tr^3)$, i.e., $E_0^3 = \langle 2,1,1 \rangle$, and finds $E_0^1 = E_0^3$ in the hash table; then we greedily match the subsequent characters of both E_0^3 and E_0^1 until the first mis-matched character (i.e., $E(Tr^3)[4] = 1$) occurs. Thus, we obtain a factor $(0,4,1)$, where 0 is the offset of $E_0^1[0]$ in $E(Tr^1)$. After that, we remove the subsequence represented by $(0,4,1)$ from $E(Tr^3)$. Since $E(Tr^3) = \emptyset$, the k-mer matching stops.

Note that k is a **pre-defined** and **fixed** value that only constrains the initial length of a matched subsequence. We adapt k-mer matching to streaming settings and detail how to select references and referentially compress trajectories in real-time.

3 TRACE framework

The framework enables the online compression and subsequent querying of streaming network-constrained trajectories (NCTs). It takes a set $\mathbf{Tr} = \{Tr^n | 1 \leq n \leq N\}$ of streaming trajectories as input. Each Tr^n contains i) the streaming path $sp(Tr^n)$ traversed by Tr^n, ii) a streaming sequence of accumulative distances $ad(Tr^n)$, and iii) a streaming time sequence $t(Tr^n)$. Figure B.3

depicts the TRACE framework that encompasses three components: a trajectory representor, a trajectory compressor, and a query processor. We deploy TRACE at the edge server and the centralized cloud layers of VEC. Raw GPS points are delivered from the smart vehicle layer to the edge server layer, where they are transformed to mapped GPS points using map-matching [35]. We do this to support smart vehicle layers with limited computational capabilities [4], thus extending the applicability. While the cost of data transmission from the smart vehicle layer to the edge server layer equals the size of the raw trajectories, this transmission is relatively efficient and scalable, because the edge server layer is geographically closer to the smart vehicle layer than to the centralized cloud layer, reducing the wireless communication energy consumption and the network load [21, 29].

Edge Server Layer. The ***trajectory representor*** converts the NCTs into speed-based and referential representations. In particular, we propose to use speed information to identify the mapped GPS point to achieve a compact representation format. Then, we apply the referential representation in [19] to exploit the similarities among subtrajectories and multiple references to achieve a high compression ratio, with the details presented in Sections 4.1 and 4.2. To select references and to represent non-references in online scenarios, we adopt k-mer matching [25]. Specifically, two hash tables are used to store the reference sets $E(\mathbf{Ref})$ and $V(\mathbf{Ref})$ for fast retrieval. Reference deletion and rewriting functions are provided that make it possible to update the tables efficiently. The former enables deleting infrequent sequences to reduce the storage cost, while the latter updates references using frequent sequences to improve the compression ratio. The detailed algorithms are covered in Sections 4.3–4.5. In the sequel, the ***trajectory compressor*** compresses the references and non-references into binary codes. A scheme for transmitting binary codes is also presented, which enables the centralized cloud to decode binary codes without extra information. The details are provided in Section 5.

Centralized Cloud Layer. The ***query processor*** is located at the centralized cloud and operates on compressed trajectories. It has an online range query algorithm that exploits indexing and filtering to achieve efficiency, to be detailed in Section 6.

4 Representation

We proceed to detail the representation of streaming trajectories.

4.1 Speed-based Representation

Existing works show that objects moving on the same road during the same time period tend to have a similar speed trend [12]. This motivates us to

Table B.2: Speed-based representation of **Tr** in Figure B.2.

n	1	2	3
$SV(Tr^n)$	44183	27444	44183
$E(Tr^n)$	$\langle 2,2,1,1,4,1,2,1,2,3,2,3\rangle$	$\langle 4,4,1,2,1,2,2,2,3,3\rangle$	$\langle 2,2,1,1,1,2,2,2,3,3,3\rangle$
$RD(Tr^n)$	0.5	0.05	0.25
$GD(Tr^n)$	$\langle 0.67, 0.5, 0.5, 0.67$ 0.33, 0.83, 0.05 $\rangle$	$\langle 0.97, 0.69, 0.35, \rangle$ $0.53, 0.11, 0.75\rangle$	$\langle 0.8, 0.5, 0.5, 0.5, 0.64\rangle$ $0.53, 0.11, 0.75, 0.67\rangle$
$V(Tr^n)$	$\langle 5, 4.76, 5, 10, \rangle$ $5.26, 20, 1.25\rangle$	$\langle 6.9, 17.11, 8.75, \rangle$ $\langle 10, 1.25, 3.75\rangle$	$\langle 5, 5, 5, 5, 8.75, \rangle$ $\langle 10, 1.25, 3.95, 7.5\rangle$
$T(Tr^n)$	$\langle$7:03:25, 0, 1, 0, 0,-1, 5, 0$\rangle$	$\langle$7:03:30, 1, -1 0,0,0,0$\rangle$	$\langle$7:03:35, 0, 0, 0, 0,0,0,0,-1,0$\rangle$

transfer the accumulative distances of trajectories to speeds and apply the referential representation to them. However, the reference speed sequence is difficult to compress substantially due to its wide range [41]. We tackle this challenge by developing the following representation.

$\boldsymbol{ad(Tr) \rightarrow (RD(Tr), GD(Tr), V(Tr))}$. We represent accumulative distances as a sequence of growth rates (denoted as $GD(Tr)$) following the first relative distance of Tr (denoted as $RD(Tr)$). The growth rate of the accumulative distance at $T(Tr)[i]$, denoted as $GD(Tr)[i]\,(i > 0)$, is calculated by $\frac{ad(Tr)[i]-ad(Tr)[i-1]}{ad(Tr)[i]-ad(Tr)[i-2]}\,(i > 1)$. Further, $GD(Tr)[1] = \frac{ad(Tr)[1]-ad(Tr)[0]}{ad(Tr)[1]}$. For example in Figure B.2a, given $ad(Tr^1)[0] = 50$, $ad(Tr^1)[1] = 150$ and $ad(Tr^1)[2] = 250$, we have $GD(Tr^1)[1] = 0.67$ and $GD(Tr^1)[2] = 0.5$. $V(Tr)$ is a sequence of speeds. The speed at $t(Tr)[i]$ is calculated by $\frac{ad(Tr)[i]-ad(Tr)[i-1]}{t(Tr)[i]-t(Tr)[i-1]}\,(i > 0)$, denoted as $V(Tr)[i]$. Given $T(Tr)$, both $GD(Tr)$ and $V(Tr)$ are able to identify $ad(Tr)$. Thus, we only store one of them, to be detailed in Section 4.2.

$\boldsymbol{sp(Tr) \rightarrow (SV(Tr), E(Tr))\ \&\ t(Tr) \rightarrow T(Tr)}$. We adopt UTCQ [19] to represent a path $sp(Tr)$ as $SV(Tr)$ followed by $E(Tr)$ and a time sequence $t(Tr)$ as $T(Tr)$. We use the UTCQ representation because it is the state-of-the-art referential compression framework for network-constrained trajectories and exhibits high compression ratios. Note that, since both $GD(Tr)$ and $V(Tr)$ enable decompressing trajectories without $T'(Tr)$ and the "0" in $E(Tr)$ used in UTCQ representation, we omit them to achieve a more compact format. Overall, the speed-based representation expresses Tr as a tuple $(SV(Tr), E(Tr), RD(Tr), GD(Tr), V(Tr), T(Tr))$. Table B.2 shows an example representation for **Tr** in Figure B.2.

4.2 Representation with Multiple-References

Based on the representation introduced in Section 4.1, we apply referential compression [19] to sub-trajectories from different streaming trajectories. Due to the use of multiple references, we modify the (S, L, M) format [19] as (ref_{id}, S, L, M), where ref_{id} is the ID of the reference. For ex-

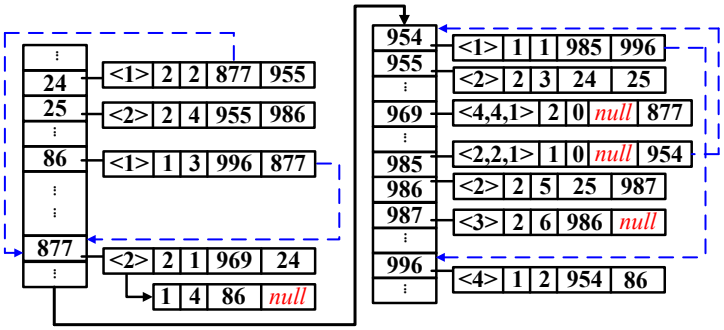


Figure B.4: A hash table $\boldsymbol{H}$ constructed according to E(Tr) in Table B.2 at 7:05:06, where $\boldsymbol{k=3}$.

ample, the referential representation of $E(Tr^3)$ w.r.t. $E(Tr^1)$ and $E(Tr^2)$ is $Com_E(Tr^3) = \langle (1,0,4,1),(2,5,5,3)\rangle$.

Different from the outgoing edge numbers, the speeds are unlikely to be exactly the same. Therefore, we consider $V(Tr^n)[i] \approx V(Tr^{n'})[i']$ if $\ddot{V}(Tr^n)[i] = \ddot{V}(Tr^{n'})[i']$, where $\ddot{V}(Tr^n)[i]$ is the integer closest to $\frac{V(Tr^n)[i]}{0.5^\eta}$ and η is the speed error bound. A larger η yields a more accurate compression at the expense of the compression ratio. If $V(Tr)$ is a non-reference, we record $GD(Tr)[i]$ if $V(Tr)[i]$ is a mis-matched value w.r.t. its reference; otherwise, we store $GD(Tr)$ instead of $V(Tr)$. This is because $V(Tr)[i]$ $(\in [0,\mu])$ can only achieve the same compression as $GD(Tr)[i]$ $(\in [0,1))$ at the cost of compression accuracy, where μ is a speed constraint of the road network. Taking $V(Tr^1)$ and $V(Tr^2)$ as references and given $\eta = 0$, we have $Com_V(Tr^3) = \langle (1,0,3,0.5),(2,2,4,0.67)\rangle$.

4.3 Reference Selection for $E(Tr^n)$

We introduce the real-time reference selection technique based on k-mer matching. Note that, we only decompose $E(Tr^n)$ and $V(Tr^n)$ into k-mers, i.e., $\phi = E$ or V, as we only apply referential representation to them.

Definition 4.1
The subsequence of $E(Tr)$ of a streaming trajectory Tr arriving at $t(Tr)[j]$ $(j > 0)$, is denoted as $E(Tr_j)$ $(j > 0)$. It starts from the edge traversed by Tr immediately after leaving the last edge of $E(Tr_{j-1})$ and ends at the edge where Tr is located at $t(Tr)[j]$.

Specifically, $E(Tr_0)$ is the outgoing edge number of $(v_s \rightarrow v_e)$ w.r.t. v_s, where $(v_s \rightarrow v_e)$ is the first edge traversed by Tr. For example, in Figure B.2a, we have $E(Tr_4^1) = \langle 4,1\rangle$ because $(v_4 \rightarrow v_5)$ is traversed by Tr^1 after leaving $(v_3 \rightarrow v_4)$ and l_4 is located on $(v_5 \rightarrow v_6)$ at $T(Tr^1)[4]$.

We construct a hash table H for $E(Tr^n)$, whose cardinality (i.e., number of hash entries) is pre-defined and fixed. Each k-mers in H is stored as a tuple $(E_i^n, n, off_i, pt_i, pd_i)$, where i) E_i^n is the i^{th} k-mer of $E(Tr^n)$; ii) n is the ID of Tr^n; iii) off_i is the offset of $E_i^n[0]$ in $E(Tr^n)$, and iv) pt_i and pd_i are the indexes of the

entries associated with E_{i-1}^n and E_{i+1}^n in H, respectively. Figure B.4 gives an example of the hash table constructed according to the streaming trajectories in Table B.2, where BKDR hashing is adopted due to its high efficiency and good distribution capability [40]. To save space, we only store $E_i^n[k-1]$ $(i > 0)$ instead of E_i^n in the hash table because the whole k-mer can be retrieved by pt_i. For example, the entry associated with $E_1^1 (= (2, 1, 1))$ in H is stored as ($\langle 1 \rangle$, 1, 1, 985, 996), where 985 and 996 are the indexes of E_0^1 and E_2^1 of Tr^1 in H, respectively.

A k-mer does not form until $|E(Tr_m^n) \cup E(Tr_{m+1}^n) \cup \cdots \cup E(Tr_{m+j}^n)| \geq k$. For example given $k = 3$, E_0^1 is formed at $t(Tr)[2]$ =7:04:06, as $E(Tr_0^1) \cup E(Tr_1^1) \cup E(Tr_2^1) = \langle 2, 2, 1 \rangle$. Once the first k-mer E_0^n of $E(Tr^n)$ is formed, we hash it to hash table H according to its hash key, denoted as *key*.

1) If $H[key] = \emptyset$, E_0^n cannot be referentially represented by the existing k-mers in H, as it is distinct from all of them. Hence, we mark $E(Tr^n)$ as a reference and create a tuple for E_0^n as well as all the subsequent k-mers E_i^n $(i > 0)$, in order to prepare for referentially representing other sequences.

2) If $H[key] \neq \emptyset$ and a tuple $(E_{i'}^{n'}, n', off_{i'}, pt_{i'}, pd_{i'})$ exists with $E_{i'}^{n'} = E_0^n$, Tr^n is marked as a non-reference. Then, we initialize a factor $E(Ma_0) = (n', off_{i'}, k, \emptyset)$ for $E(Tr^n)$, where k is the current matched length and $\emptyset$ indicates that the mis-matched value of $E(Ma_0)$ is unknown. Next, we follow an existing work [25] to greedily match the subsequent character $E_{i+1}^n[k-1]$ $(i > 0)$ with that in $E(Tr^{n'})$. This process ends when a mis-matched character M occurs. So far, a factor (n', S, L, M) is generated, where L is the final matched length of the subsequence of $E(Tr^{n'})$ and $S = off_{i'}$. Then we wait until another k-mer forms and repeat the process. Clearly, the time delay occurs mainly when waiting for k mapped GPS points before initializing a factor.

Example 4.1
Following the example shown in Figure B.4 and Table B.2, since E_0^3 matches with E_0^1, we initialize a factor $(1, 0, 3, \emptyset)$ for it. For the arriving $E(Tr_3^3) = \langle 1 \rangle$, we retrieve $E(Tr^1)[3]$ according to the index pd_0 associated with E_0^1, i.e, 954, and compare it with $E(Tr_3^3)$ $(= E_1^3[2])$. After that, we update the factor to $(1, 0, 4, \emptyset)$ due to $E(Tr^1)[3] = E(Tr_3^3)$. As $E(Tr^1)[4] \neq E_2^3[2]$, we generate a factor, i.e., $(1, 0, 4, 1)$, for $Com_E(Tr^3)$. Then, we wait until $E(Tr_6^3)$ arrives due to $|E(Tr_5^3) \cup E(Tr_6^3)| \geq 3$ and repeat the process. Finally, we get $Com_E(Tr^3) = \langle (1, 0, 4, 1), (2, 5, 5, 3) \rangle$.

However, a streaming trajectory Tr^n can be a non-reference at the beginning but cannot be referentially represented since a timestamp t_i, because its k-mer formed at t_i cannot match any tuple stored in the hash table H. In this case, we call Tr^n as a *hy-reference* and create a tuple for each k-mer of Tr^n generated since t_i to referentially represent other sequences. Intuitively, the former part of a hy-reference Tr^n is a non-reference that is referentially represented, while

the latter part of it is a reference.

We store a repetitive E_i^n as (n, off_i, pt_i, pd_i) by omitting the redundant E_i^n. We associate this tuple with $(E_{i'}^{n'}, n', off_{i'}, pt_{i'}, pd_{i'})$ ($E_{i'}^{n'} = E_i^n$) (i.e., they are stored in the same hash entry) to trace back the subsequence of Tr^n during k-mer matching. As shown in Figure B.4, the tuple of E_4^1, i.e., $(1, 4, 86, null)$ is associated with that of E_1^2, as $E_4^1 = E_1^2$. Here, *null* indicates that E_5^1 is unknown at the current timestamp. Moreover, E_i^n and $E_{i'}^{n'}$ may be stored in the same hash entry even if $E_i^n \neq E_{i'}^{n'}$ due to hash collisions. According the introduction of k-mer matching in Section 2.3, k remains unchanged during compression and only determines the length of the initial matched subsequence. However, k cannot be too large or too small. A too large k may result in a very high probability of failed matching and thus excessively many references, and a too small k may lead to many "trivial" matches, where each factor of a non-reference represents a very short subsequence. We study the impact of k on compression performance in Section 7.

4.4 Reference Deletion for $E(Tr^n)$

The number of k-mers stored in the hash table increases over time. To reduce the storage cost, we delete the k-mers corresponding to references from the hash table that have not been visited for a long time. A reference $E(Ref)$ is **visited** if i) it referentially represents a non-reference or ii) its corresponding data is still arriving. We define G_o as the set of the references at timestamp t_o. Specifically, $G_o[i]$, that represents a reference $E(Ref)$, denotes the tuple $(ref_{id}, G_o[i].tl)$, where ref_{id} is the ID of the reference $G_o[i]$, and $G_o[i].tl$ is the timestamp when $G_o[i]$ was most recently visited.

Definition 4.2
A reference $G_o[i]$ is **outdated** at t_o if its **freshness** at t_o, denoted as $G_o[i].f$, satisfies $G_o[i].f < C \cdot \frac{F_o}{|G_o|}$, where $F_o = \sum_{i'=0}^{|G_o|-1} G_o[i'].f$ and $C\,(0 < C \leq 1)$ is the deletion coefficient.

We set the deletion coefficient $C \in (0, 1]$ because we consider a reference as outdated only if its freshness is below the average freshness $\frac{F_o}{|G_o|}$. A larger C implies that references expire more easily. We calculate $G_o[i].f$ as follows:

$$G_o[i].f = \lambda^{t_o - G_o[i].tl} \quad (t_o \geq G_o[i].tl), \tag{B.1}$$

where $\lambda \in (0, 1)$ is a decay factor [7]. In Figure B.2, given the current timestamp t_o =7:03:26 and $\lambda = 0.998$, we get $E(Tr^1).f = 0.998$ at t_o.

Definition 4.2 is more effective than using a fixed threshold to determine whether a reference is expired. Specifically, using a fixed threshold, if no trajectory arrives for a long time due to an occasional interrupt of communications, all references in the hash table will expire. A naive solution is to update the

freshness of each reference in the hash table at each timestamp in order to compute the average freshness and then to identify the outdated data. To improve efficiency, we propose to compute the freshness of parts of $G_o[i]$ $(0 \leq i < |G_o|)$ at t_o. We sort G_o in ascend order of $G_o[i].f$ $(0 \leq i < |G_o|)$. Then, we detect expired data in order until $G_o[i'].f \geq C \cdot \frac{F_o}{|G_o|}$ $(0 < i' < |G_o|)$. Specifically, no k-mers are deleted if $G_o[0].f \geq C \cdot \frac{F_o}{|G_o|}$. This process can actually be realized without sorting, as shown in Algorithm 2. Next, we define a set containing the references in G_o that are visited at t_o, denoted as Rv_o $(Rv_o \subseteq G_o)$. Given $F_{o'}$, F_o $(t_o > t_{o'})$ is updated as follows:

$$F_o = (F_{o'} - \sum_{G_{o'}[i] \in Rv_o} G_{o'}[i].f) \cdot \lambda^{t_o - t_{o'}} + |Rv_o| \tag{B.2}$$

Formula B.2 enables us to update F_o by only computing the freshness of $G_{o'}[i]$, such that $G_{o'}[i] \in Rv_o$. The derivation of it is omitted due to the space limitation.

Algorithm 2 details the reference deletion. Assuming that the latest timestamp when new data arrives is $t_{o'}$. The algorithm searches each $G_{o'}[i] \in Rv_o$ in the current reference set $G_{o'}$ and removes $G_{o'}[i]$ from $G_{o'}$ if it exists in Rv_o (Lines 1–4). Then, each reference $E(Ref) \in Rv_o$ is appended to $G_{o'}$ (Line 6). This way, it sorts $G_{o'}$ in ascending order of freshness. Meanwhile, Formula B.2 is applied to calculate F_o (Lines 5 and 7) and $G_{o'}$ is updated to G_o. After "sorting", we check the freshness of references in G_o by starting from its first reference every time, in order to only calculate the freshness of the potentially outdated references (Lines 8–11). Finally, the updated references G_o and the sum of their freshness F_o are returned (Line 12). This approach reduces the time complexity of updating freshness from $O(|G_o|)$ to $O(|Rv_o| + |EX_o|)$, where EX_o is the set of expired references at t_o. We evaluate the effectiveness and efficiency of reference deletion via experiments in Section 7.

4.5 Reference Rewriting for $E(Tr^n)$

To further improve the compression ratio, we rewrite the references $E(Ref)$ in real-time. The motivation is that a non-frequent edge sequence may become a reference if it arrives early. For example, given a frequent path $\langle 44183, 2, 2, 1, 1, 1, 2, 2\rangle$, $\langle 2, 2, 1, 1, 1, 2, 2\rangle$, i.e., a subsequence of $E(Tr^3)$, should be stored as k-mers. However, since Tr^1 arrives before Tr^3 and $E(Tr^1)$ can referentially represent $E(Tr^3)$, $E(Tr^3)$ is made as a non-reference and Tr^1 is a reference, as shown in Example 4.1. In this case, for another new arriving subsequent Tr^n that also traverses $\langle 44183, 2, 2, 1, 1, 1, 2, 2\rangle$, $Com_E(Tr^n)$ will contain at least two factors. To address the problem, we detect the frequent subsequence and rewrite the k-mers in real-time. Following an existing study [31], we define the rewriting candidate below.

Algorithm 2: Reference Deletion Algorithm

Input: the reference set $G_{o'}$ at $t_{o'}$, the set Rv_o, the sum of freshness $F_{o'}$ and a threshold C

Output: the reference set G_o and the sum of freshness F_o

for *each reference* $E(Ref) \in Rv_o$ **do**
  if $E(Ref) \in G_{o'}$ *with* $E(Ref)$=$G_{o'}[i]$ **then**
    compute $G_{o'}[i].f$ using Formula B.1
    remove the tuple $(ref_{id}, G_{o'}[i].tl)$ from $G_{o'}$
    $F_{o'} \leftarrow F_{o'} - G_{o'}[i].f$
  append the tuple (ref_{id}, t_o) to $G_{o'}$
$F_o \leftarrow F_{o'} \cdot \lambda^{t_o - t_{o'}} + |Rv_o|$
while $\lambda^{t_o - G_{o'}[0].tl} < C \cdot \frac{F_o}{|G_{o'}|}$ **do**
  $F_o \leftarrow F_o - \lambda^{t_o - G_{o'}[0].tl}$
  remove the tuple $(ref_{id}, G_{o'}[0].tl)$ from $G_{o'}$
  delete the k-mers associated with $G_{o'}[0]$ from the hash table
return $G_{o'} (= G_o)$ *and* F_o

Definition 4.3
A **rewriting candidate** is a reference $E(Ref)$ that represents a non-reference $E(Nref)$ as $Com_E(Nref) = \langle \cdots, (ref_{id}, S, L, M), (ref_{id}, S', L', M'), \cdots \rangle$, where $S + L + 1 = S'$ and ref_{id} is the ID of the reference Ref.

Given a rewriting candidate Ref, we can **merge** two factors (ref_{id}, S, L, M) and (ref_{id}, S', L', M') into one factor $(ref_{id}, S, L + 1 + L', M')$, by replacing $E(Ref)[S + L]$ with M, which is called a rewriting operation.

Example 4.2
Given $E(Tr^4) = \langle 2, 2, 1, 1, 4, 1, 3, 1, 2, 3, 3 \rangle$ such that Tr^4 arrives after Tr^1 ends, $E(Tr^1)$ becomes a rewriting candidate as $Com_E(Tr^4) = \langle (1, 0, 6, 3), (1, 7, 3, 3) \rangle$. If $E(Tr^1)[6]$ is replaced with 3, $Com_E(Tr^4)$ will only contain one factor, i.e., $(1, 0, 10, 3)$, which leads to a higher compression. Similarly, given $E(Tr^5) = \langle 1, 1, 4, 1, 3, 1, 2, 3, 3 \rangle$ such that Tr^5 also arrives after Tr^1 terminates, the compression can be improved by replacing $E(Tr^1)[6]$ with 3.

We construct an array rp for each $E(Ref)$, where $rp[i]$ is a list recording M corresponding to a rewriting operation for $E(Ref)[i]$ $(1 \leq i < |E(Ref) - 1|)$ and its frequency of occurrence, denoted as $f(M)$. Following Example 4.2, $rp[6]$ of $E(Tr^1)$ is $\langle (3, 2) \rangle$, due to $M = 3$ and $f(M) = 2$. Obviously, a prerequisite for conducting a rewriting operation is that $f(M)$ should be large. This is to guarantee that a frequent subsequence (i.e., a frequent path) is generated by the rewriting. Moreover, we should ensure that the factors overlapping $E(Ref)[i]$ occur rarely. Given a factor (ref_{id}, S, L, M), it intersects $E(Ref)[i]$, if ref_{id} is the ID of Ref and $S \leq i < S + L$.

Algorithm 3: Reference Rewriting Algorithm

Input: a rewriting operation, i.e., replacing $E(Ref)[i]$ with M, a $b \times b$ factor matrix **FA** of $E(Ref)$, an array rp of $E(Ref)$ and a threshold α
Output: a rewritten reference or $\emptyset$

$f(M) \leftarrow f(M) + 1$, where $f(M)$ is associated with $rp[i]$
if $\forall M' \in rp[i]\,(f(M) \geq f(M')) \wedge f(M) \geq \alpha$ **then**
  if $\sum_{x=1}^{\lfloor \frac{i}{k} \rfloor + 1} \sum_{y=\lfloor \frac{i}{k} \rfloor + 1}^{b} FA^{xy} < f(M)$ **then**
    replace $E(Ref)[i]$ with M
    update k-mers associated with $E(Ref)$
    delete **FA** and rp of $E(Ref)$
    return *a rewritten reference*
  else
    return $\emptyset$ /* no operation will be conducted */
else
  return $\emptyset$ /* no operation will be conducted */

Example 4.3
Given $E(Tr^6) = \langle 2,2,1,1,4,1,2,1,2,2,2,2,1,3\rangle$ such that Tr^6 arrives after Tr^1 terminates, we get $Com_E(Tr^6) = \langle (1,0,9,2),(1,0,3,3)\rangle$, where $(1,0,9,2)$ intersects $E(Tr^1)[6]$. Following Example 4.2, even if replacing $E(Tr^1)[6]$ with 3 reduces the number of factors of both $Com_E(Tr^4)$ and $Com_E(Tr^5)$, it produces one more factor for $Com_E(Tr^6)$. This indicates that we should rewrite $E(Tr^1)[6]$ only when it does not frequently intersect any factors.

The above analysis implies that we should record each factor that intersects $E(Ref)[i]$ $(1 \leq i < |E(Ref)| - 1)$ for rewriting. Intuitively, this results in a high space-time consumption. Inspired by the regular square grid graph [10], we construct a $b \times b$ factor matrix **FA** for each $E(Ref)$, where $b = \lceil \frac{|E(Ref)|_r}{k} \rceil$. Here, $|E(Ref)|_r$ is the length of the subsequence of $E(Ref)$ used as a reference. $FA[x][y]$ $(x, y > 0)$, denoted as FA^{xy}, is associated with a subsequence of $E(Ref)$ in its factor matrix, i.e., $\langle E(Ref)[(x-1)\cdot k], \cdots, E(Ref)[y \cdot k - 1]\rangle$, and counts the frequency of factors (ref_{id}, S, L, M) intersecting $E(Ref)[i]$ $((x-1)\cdot k \leq i \leq y \cdot k - 1)$, where ref_{id} is the ID of Ref. Thus, we update **FA** once a factor is generated.

Proposition 4.1
A factor (ref_{id}, S, L, M) contributes to FA^{xy}, where $x = \lfloor \frac{S}{k} \rfloor + 1$ and $y = \lceil \frac{S+L}{k} \rceil$.

Example 4.4
Following Examples 4.1, 4.2, and 4.3 and assuming that $\mathbf{Tr} = \{Tr^1, Tr^2, Tr^3, Tr^4,$

$Tr^5, Tr^6\}$, the factor matrix **FA** of $E(Tr^1)$ is:

$$\mathbf{FA} = \begin{bmatrix} [0,2] & [0,5] & [0,8] & [0,11] \\ & [3,5] & [3,8] & [3,11] \\ & & [6,8] & [6,11] \\ & & & [9,11] \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 3 & 1 & 0 \\ & 0 & 0 & 0 \\ & & 0 & 2 \\ & & & 0 \end{bmatrix}$$

Here, the left part of **FA** intuitively gives the subsequences of $E(Tr^1)$ corresponding to FA^{xy}, and the right part shows the value of FA^{xy}. For instance, $FA^{12} = 3$ is contributed by three factors, i.e, $(1,0,4,1)$, $(1,0,6,3)$, and $(1,2,4,3)$.

Lemma 4.1
Given a factor (ref_{id}, S, L, M) that intersects $E(Ref)[i]$ and a $b \times b$ factor matrix **FA** of $E(Ref)$, (ref_{id}, S, L, M) can only contribute to FA^{xy}, where $x \leq \frac{i}{k} + 1 \wedge y > \frac{i}{k}$.

Proof. As (ref_{id}, S, L, M) intersects $E(Ref)[i]$, we have $S \leq i < S + L$. Proposition 4.1 guarantees that $S \geq (x-1) \cdot k \wedge S + L \leq y \cdot k$ if (ref_{id}, S, L, M) contributes to FA^{xy}. Hence, we have $i \geq (x-1) \cdot k \wedge i < y \cdot k$, i.e., $x \leq \frac{i}{k} + 1 \wedge y > \frac{i}{k}$. □

Lemma 4.2
Given a $b \times b$ factor matrix **FA** of $E(Ref)$, the maximum number of factors that intersect $E(Ref)[i]$ is $\sum_{x=1}^{\lfloor \frac{i}{k} \rfloor + 1} \sum_{y=\lfloor \frac{i}{k} \rfloor + 1}^{b} FA^{xy}$.

Proof. Following Lemma 4.1, the factors intersecting $E(Ref)[i]$ can only contribute to FA^{xy}, where $x \leq \frac{i}{k} + 1 \wedge y > \frac{i}{k}$. As a result, the maximum number of the factors that intersect $E(Ref)[i]$ is $\sum_{x=1}^{\lfloor \frac{i}{k} \rfloor + 1} \sum_{y=\lfloor \frac{i}{k} \rfloor + 1}^{b} FA^{xy}$. □

Example 4.5
Continuing Example 4.4, the maximum number of the factors intersecting $E(Tr)[6]$ is $\sum_{x=1}^{3} \sum_{y=3}^{4} FA^{xy} = 3$, where only 6 out of 10 elements in FA need to be visited.

Based on the above conclusions, we present the conditions for implementing a rewriting operation, i.e, replacing $E(Ref)[i]$ with M: i) $f(M) \geq \alpha$, where $\forall M' \in rp[i]$ $(f(M) \geq f(M'))$ and α (≥ 1) is the rewriting coefficient, and ii) $\sum_{x=1}^{\lfloor \frac{i}{k} \rfloor + 1} \sum_{y=\lfloor \frac{i}{k} \rfloor + 1}^{b} FA^{xy} < f(M)$. The first condition ensures that M occurs more frequently than other characters for a given position i. The second condition ensures that the maximum number of factors intersecting $E(Ref)[i]$ is smaller than $f(M)$. Note that a smaller α means that $f(M) \geq \alpha$ occurs more often, thus implying more frequent rewriting.

Algorithm 3 presents the pseudo-code of rewriting references. If a reference is rewritten, we update its corresponding k-mers (Line 5). Moreover, we

store the rewritten reference by referentially representing it according to the corresponding original one. The aim is to reduce the storage needed when introducing a new reference. Thus, we do not consider to rewrite a reference more than once, as it introduces θ-order compression ($\theta > 2$) at the cost of decreased efficiency of decompression and querying. This is also the reason that we delete **FA** and *rp* after rewriting a reference (Line 6). In addition, we observe that the factor matrix **FA** of $E(Ref)$ still takes up unnecessary space due to FA^{xy} $(y > x) = \emptyset$. Hence, to store FA^{xy}, we construct a vector FV of size $\frac{b\cdot(b+1)}{2}$, where $b = \lceil \frac{|E(Ref)|_r}{k} \rceil$ and $|E(Ref)|_r$ is the length of $E(Ref)$ that is used as a reference. FV is enlarged over time according to the latest $|E(Ref)|_r$. The worst case time complexity of reference rewriting for a reference is $O(|FV|)$. However, Lemma 4.2 enables rewriting by scanning part of FV (as shown in Example 7), which enhances the efficiency of reference rewriting. This is also studied in Section 7.

4.6 Reference Selection and Deletion for $V(Tr^n)$

The reference selection for $V(Tr^n)$ is almost the same as that for $E(Tr^n)$. However, as a speed $V(Tr^n)[i]$ is a float, we cannot directly apply k-mer matching to it. Instead, we convert it to an integer $\dot{V}(Tr^n)[i]$, where $\dot{V}(Tr^n)[i]$ is the integer closest to $\frac{V(Tr^n)[i]}{0.5^\eta}$. This strategy is consistent with the error-bounded referential representation in Section 4.2. The reference deletion for $V(Tr^n)$ is also similar to that for $E(Tr^n)$; thus, we omit it. Moreover, we do not rewrite a reference $V(Ref)$, as speed patterns may vary substantially during different time periods [12, 24] and the latest patterns have already been mined by reference selection.

5 Compression

We proceed to present the binary encoding and data transmission of streaming trajectories.

5.1 Binary Encoding

We denote the binary code of *seq* as *sêq*. e.g., the binary code of $E(\cdot)$ is denoted as $\hat{E}(\cdot)$.

Binary Encoding of References. We follow UTCQ [19] to compress $E(Ref)$. Moreover, we adapt a typical scheme [41] to encode $RD(Ref)$ and $GD(Ref)$. Specifically, given an encoding error bound γ, the binary code $\hat{fv}$ of a floating number fv is calculated as $\hat{fv} = \arg\min_{\hat{fv}_m} \left| \sum_{i=1}^{|\hat{fv}_m|} \hat{fv}_m[i-1] \cdot \frac{1}{2^i} - fv \right|$, where $|\hat{fv}_m| = \gamma$. For example, given $fv = 0.37$ and $\gamma = 3$, we get $\hat{fv}$=011, i.e., fv is

approximated as 0.375. The binary code of a reference $\phi(Ref)$ ($\phi = E, GD$) is delivered to the centralized cloud immediately after it is generated.

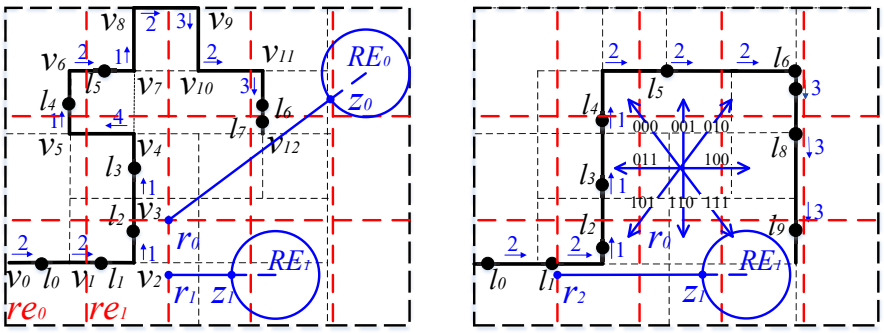
Binary Encoding of Non-references. We set the lengths of the binary codes of both S and L the same, denoted as len, and record it for decoding a factor (ref_{id}, S, L, M) in a streaming setting. The Exp-Golomb encoding [38] is adopted to compress len. Moreover, len is set to 0 if the reference $\phi(Ref)$ has terminated before generating a factor (ref_{id}, S, L, M). This way, we can improve compression as $len = 0$ only takes 1 bit, and both $\hat{S}$ and $\hat{L}$ are still decodable as the length of $\phi(Ref)$ is known [19]. Finally, $\hat{M}$ takes $\lceil \log_2 o \rceil$ bits for $Com_E(Nref)$ and takes γ bits for $Com_V(Nref)$, where o is the maximum number of outgoing edges for any vertex $v \in V$.

5.2 Transmission of Compressed Binary Codes

As illustrated in Section 4.3, a factor (ref_{id}, S, L, M) cannot be generated until the mis-matched character M is found. It is easy to recognize each part of a factor if we transmit it as a whole to the centralized cloud. However, in this case, the centralized query processor is unable to receive the latest data in real-time, resulting in inaccurate results. To avoid this, we continue transmitting the up-to-date information of a factor during its formation.

We propose a data transmission strategy, which targets low transmission overhead and enables decoding at the centralized cloud without the need of delivering extra information. Given the value of k for k-mer, the transmission of a factor (ref_{id}, S, L, M) is completed in three states: ① transferring the initialized factor $(ref_{id}, S, L', \emptyset)$, where $L' \geq k$ and $|\hat{S}| = |\hat{L'}|$; ② transferring the updated L'; and ③ transferring the mis-matched element M when L' is updated to L. We denote the binary code of a factor transmitted to the centralized cloud at each timestamp as $\hat{bc}$. Obviously, $|\hat{bc}| = |\hat{ref_{id}}| + 2 \cdot |\hat{S}| \geq |\hat{ref_{id}}| + 2 \cdot k$ at step ①, and the $\hat{bc}$ that is used to update L' at step ② is always $|\hat{ref_{id}}| + 1$. As the maximum number of outgoing edges for any vertices in a road network is generally no less than 4, i.e, $o \geq 4$, and we set $\gamma \geq 2$, we have $|\hat{bc}| = |\hat{ref_{id}}| + |\hat{M}| > |\hat{ref_{id}}| + 1$ at step ③. If we let $2 \cdot |\hat{S}| \neq |\hat{M}|$, the binary codes transferred during the above three states can be distinguished just by $|\hat{bc}|$. The initial state is ③.

Example 5.1
Continuing Example 4.3 and letting $\hat{ref_{id}}$ take 3 bits, the first factor of $Com_E(Tr^6)$ is initialized as $(1, 0, 3, \emptyset)$ and thus is encoded as $(000, 00, 10)$, i.e., $\hat{bc} = 0000010$. Then it is sent to the centralized cloud, which triggers the state transition from ③ to ①. Next, we continue to transfer $\hat{bc} = 0001$ before the mis-matched value $M = 2$ is found, during which the state is first transferred to ② and then remains unchanged. Meanwhile, L' continues to be incremented by 1. Once the centralized cloud receives $M = 2$, i.e., $\hat{bc} = 000001$, a factor is generated and stored in the form $(\hat{M}, \hat{len}, \hat{ref_{id}}, \hat{S}, \hat{L})$, i.e., $(001, 0, 000, 0000, 1000)$,



(a) Tr^1 **(b)** Tr^3

Figure B.5: The partition of the road network G in Figure B.2.

in the centralized cloud, where we assume $o = 7$, i.e., $\hat{M}$ takes 3 bits.

6 Query Processing

6.1 Query Definition

Definition 6.1
Given a query region RE and a set of compressed streaming trajectories $\hat{\boldsymbol{Tr}}$, a **range query range**$(\hat{\boldsymbol{Tr}}, RE)$ returns the set of streaming trajectories $Tr^n (1 \leq n \leq N)$ in $\boldsymbol{Tr}$, such that $Tr^n \cap RE \neq \emptyset$ at the current timestamp.

We denote the lastest timestamp of a trajectory Tr^n as $Tr^n.tp$, i.e., $Tr^n.tp = t(Tr^n)[|t(Tr^n)| - 1]$.

Example 6.1
Considering the streaming trajectories in Figure B.2 and assuming that the current timestamp is 7:04:06, we have $Tr^1.tp$ =7:04:06 and $Tr^3.tp$ =7:03:55. Given $k = 2$, $\gamma = 7$, $\eta = 0$, and $|(v_0 \rightarrow v_1)| = |(v_1 \rightarrow v_2)| = |(v_2 \rightarrow v_3)| = 100$, we get $ad(Tr^1)[2] = 254.76$, $ad(Tr^3)[1] = 127.38$, and $V(Tr^3)[1] = 5.12$ after decompression, i.e., Tr^1 is located on $(v_2 \rightarrow v_3)$ and Tr^3 is located on $(v_1 \rightarrow v_2)$; thus, the range query $\boldsymbol{range}(\hat{\boldsymbol{Tr}}, RE_1)$ returns $\emptyset$.

A naive strategy for computing range queries over compressed streaming trajectories is to decompress each $Tr^n (1 \leq n \leq N)$ and calculate their current locations, which is time-consuming. Instead we introduce indexing and filtering to achieve fast query processing.

6.2 Index and Filtering Technique

We partition the road network G using grid cells re_m, each of which links to the streaming trajectories that are currently located in it. The number of grid

cells is denoted by gc. Figure B.5 partitions the road network G in Figure B.2. Considering the examples in Figure B.2 and assuming the current timestamp is 7:03:25, re_0 links to Tr^1.

Definition 6.2
The **minimum distance** $mind(re_m, RE)$ between a grid cell re_m and a query region RE is the distance between a location r and a location z, denoted as $|rz|$, where $r \in re_m \wedge z \in RE \wedge \forall r' \neq r\ (r' \in re_m \wedge |r'z| \geq |rz|) \wedge \forall z' \neq z\ (z' \in RE \wedge |rz'| \geq |rz|)$.

For instance in Figure B.5a, the minimum distance between grid cell re_1 and query region RE_0 is $|r_0z_0|$. Since queries are computed centrally, we need to calculate the current location of each streaming trajectory from the most recently arrived data. Given a speed constraint μ of the road network and a timestamp tc, the **reachable distance** of a streaming trajectory Tr^n w.r.t. tc, denoted as dis, is $(tc - Tr^n.tp) \cdot \mu$. Following Example 6.1 and given $\mu = 21$, the reachable distance of Tr^1 w.r.t. 7:04:06 is $0 \times 21 = 0$, while that of Tr^3 w.r.t. 7:04:06 is $11 \times 21 = 231$.

Lemma 6.1
Given a range query $\mathbf{range}(\hat{\boldsymbol{Tr}}, RE)$, the current timestamp tc, the reachable distance dis of Tr^n w.r.t. tc, and Tr^n located in grid cell re_m at $Tr^n.tp$, if the minimum distance $mind(re_m, RE) > dis$, Tr^n cannot be in the result.

Proof. Assuming that Tr^n has a mapped location l at $Tr^n.tp$, we have $l \in re_m$ as Tr^n located in grid cell re_m at $Tr^n.tp$. Since $mind(re_m, RE) > dis$, the distance between l and RE must also exceed the reachable distance dis of Tr^n w.r.t. tc, i.e., Tr^n cannot reach RE at tc. Hence, Tr^n cannot be in the query result. □

Lemma 6.1 enables pruning Tr^n without computing its location at the current timestamp. Following Example 6.1 and given $|r_2z_1| = 233$ in Figure B.5b, we do not need to decompress $\hat{Com}_E(Tr^3)$ and $\hat{Com}_V(Tr^3)$ if a range query $\boldsymbol{range}(\hat{\mathbf{Tr}}, RE_1)$ arrives at 7:04:06. This is because Tr^3 overlaps re_0 at 7:03:55 and the reachable distance of Tr^3 w.r.t. 7:04:06 is 231<233. If Tr^n cannot be filtered, we need to fully decompress it to calculate its current location. The details are omitted due to the space limitation.

6.3 Index Transmission

Index information is created at the edge server once new data arrives, while query processing occurs centrally. Hence, we need to deliver the index to the centralized cloud. As illustrated in Section 5.2, we always transmit a compressed trajectory once it is (referentially) represented, in order to support accurate queries. A naive strategy is to transfer the ID of the grid cell where Tr^n is located at each timestamp, which incurs high transmission cost.

We propose to transfer the ID of the grid cell, denoted as g_{id}, for Tr^n only when it changes. As shown in Figure B.5b, Tr^n can enter at most 8 grid cells if

it leaves the current grid cell. Therefore, $\hat{g_{id}}$ takes 3 bits and is appended to $\hat{bc}$. We add one bit at the beginning of $\hat{bc}$ to identify whether it carries index information. The transmission algorithm that includes indexes is very similar to that in Section 5.2, so we omit the details due to the space limitation.

6.4 Discussion

TRACE is able to adopt and support a variety of partitioning methods and queries, as discussed next.

Road Network Partition. We introduce two representative partitioning methods, spatial partitioning [20, 30] and graph-based partitioning [1, 36, 37], which are alternatives to our grid partitioning. Quad-tree partitioning [20, 30] is used often in different settings. It recursively decomposes the space while considering the spatial distribution of the underlying data, and it stops when some pre-defined conditions are satisfied. Each tree node represents a subregion and has either exactly four children (an internal node), or no children (a leaf node). In congestion-based partitioning of a road network [1, 36, 37], edges are associated with feature values and traffic densities. Based on this information, different heterogeneous subregions of a road network are identified that exhibit homogeneous traffic congestion patterns internally.

The above-mentioned partitioning methods and corresponding indexing techniques can be adapted straightforwardly to TRACE, by considering subregions as grid cells. We use grid partitioning because it is simple, is very efficient, and has low construction cost [24]. However, exploration of possible benefits of other partitioning techniques is a relevant topic for future work.

Query. We introduce two different types of queries that can be supported by TRACE, i.e., a shortest path query and a KNN query. We use l_c to denote the mapped GPS point of a streaming trajectory Tr^n at the current timestamp t_c.

Shortest path query: ***short***$(\hat{Tr^n}, v_q)$. Given a vertex v_q and a compressed streaming trajectory $\hat{Tr^n}$, ***short***$(\hat{Tr^n}, v_q)$ returns the shortest path distance between l_c and v_q. To answer ***short***$(\hat{Tr^n}, v_q)$, we first decompress $\hat{T}(Tr^n)$, $\hat{E}(Tr^n)$ and $\hat{GD}(Tr^n)$ (or $\hat{V}(Tr^n)$) to get $l_c = ((v_i \rightarrow v_j), nd(v_i, l_c), t_c))$. Then the shortest path distance between l_c and v_q is obtained by computing the shortest path distances between v_i and v_q and between v_j and v_q [15]. This process can be facilitated by constructing a G*-tree [23].

KNN query: ***KNN***$(\hat{Tr^n}, \hat{\mathbf{Tr}},)$. Given a threshold , a compressed streaming trajectory $\hat{Tr^n}$, and a set of compressed streaming trajectories $\hat{\mathbf{Tr}}$, ***KNN***$(\hat{Tr^n}, \hat{\mathbf{Tr}},)$ returns the top streaming trajectories in $\hat{\mathbf{Tr}}$ ranked by their shortest path distances to l_c at t_c in ascending order. The process of computing ***KNN***$(\hat{Tr^n}, \hat{\mathbf{Tr}},)$ is similar to that of computing ***short***$(\hat{Tr^n}, v_q)$. In addition, a priority queue is maintained to perform the most promising vertex expansions [23].

Table B.3: Trajectory datasets.

Datasets	Storage of NCTs	# of NCTs	# of edges per NCT
Denmark	3.47 GB	415,920	Average 95.844
Hangzhou	24.60 GB	1,918,677	Average 250.540
Synthetic	384.61 GB	50,000,000	Average 137.712

Table B.4: Parameter ranges and default values.

Parameter	Range
the length of k-mer	5, 7, **9**, 10, 11, 13, **15**, 20, 25
the value of C	0.1, 0.3, 0.5, 0.7, **0.9**
the value of α	**2**, 3, 4, 5, **6**, 8, ∞
the number of grid cells gc	8^2, 16^2, 32^2, $\mathbf{64^2}$, 128^2

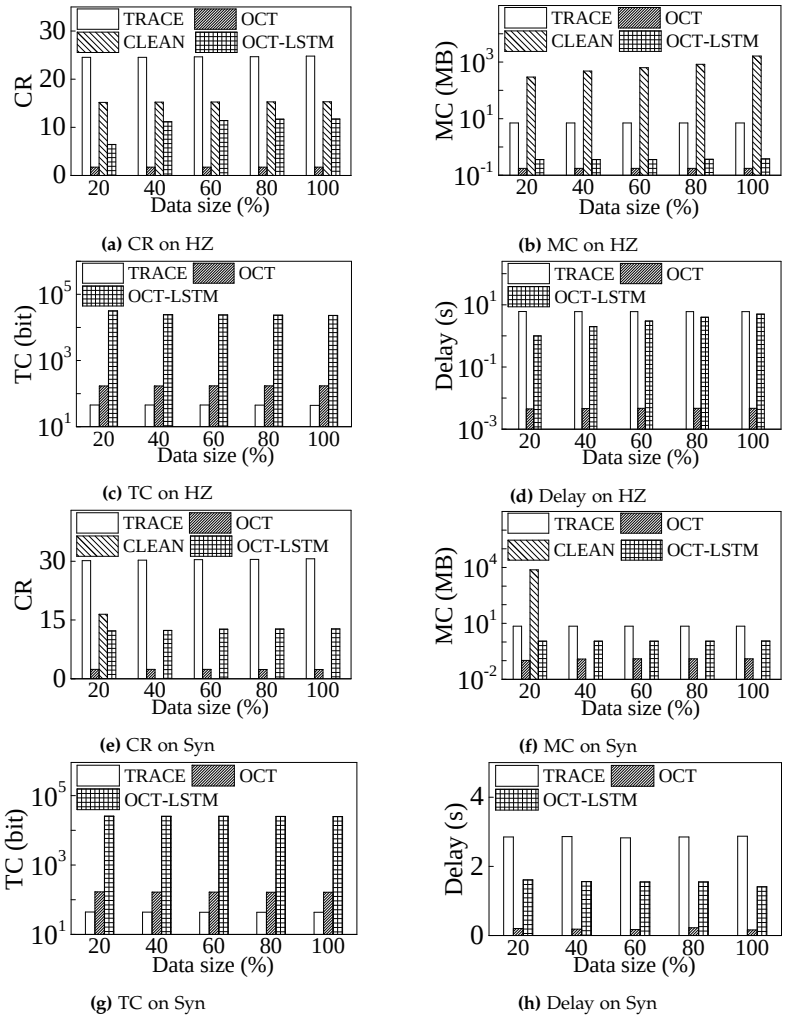
7 Experimental Evaluation

7.1 Experimental Setting

Datasets. We use two real datasets, Denmark (DK) and Hangzhou (HZ), and a synthetic dataset, Synthetic (Syn), as described in Table B.3. DK is collected from 162 vehicles over about 2 years from Jan. 2007 to Dec. 2008 in Denmark, while HZ is collected from 24,515 taxis during Nov. 2011 in Hangzhou, China. Syn is generated using the road network of Hangzhou. It contains five data groups, each with the same number of trajectories (i.e., Syn0.1, Syn0.3, Syn0.5, Syn0.7, and Syn0.9), where the similarities between each pair of trajectories are 0.1 ± 0.05, 0.3 ± 0.05, 0.5 ± 0.05, 0.7 ± 0.05, and 0.9 ± 0.05, respectively. The similarity is measured according to the Longest Common Road Segment (LCRS) [42]. We generate similar speed patterns when trajectories traverse an LCRS. Specifically, any two subsequences of speeds corresponding to an LCRS, denoted as $sub(V^n)$ and $sub(V^{n'})$, satisfy $|sub(V^n)| = |sub(V^{n'})| \wedge |sub(V_i^n) - sub(V_i^{n'})| \leq \frac{1}{2}^{\eta}$ $(0 \leq i < |sub(V^n)|)$, where $\eta = 7$. The default sample interval of HZ and Syn is 20s, and that of DK is 1s.

Parameter Setting. In the experiments, we study the effect on the performance of the parameters summarized in Table B.4. We set λ to 0.998 and γ to 3 on both datasets, and set η to 3 on DK and 7 on HZ, respectively. Note that the default values of parameters for Syn are the same as those for HZ. The cardinalities of the hash tables for storing $E(\mathbf{Ref})$ and $V(\mathbf{Ref})$, are both 1000. All algorithms are implemented in C++ and run on a computer with an Intel Core i9-9880H CPU (2.30 GHz) and 32 GB memory.

Comparison Algorithms. We compare TRACE with three methods: CLEAN [44], OCT-LSTM [5], and OCT [5]. CLEAN is an offline method, while OCT-LSTM trains an LSTM model to obtain repetitive patterns of time-distance sequences using historical data. OCT-LSTM compresses time-distance sequences by discarding data if the prediction deviation is smaller than an error bound. OCT [5] is similar to OCT-LSTM, except that it uses a linear model for



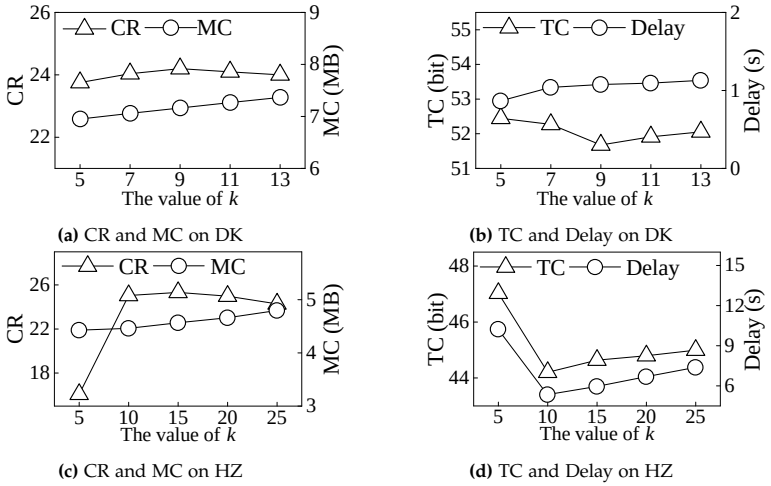
(a) CR on HZ **(b)** MC on HZ

(c) TC on HZ **(d)** Delay on HZ

(e) CR on Syn **(f)** MC on Syn

(g) TC on Syn **(h)** Delay on Syn

Figure B.6: Comparison and scalability.



(a) CR and MC on DK **(b)** TC and Delay on DK

(c) CR and MC on HZ **(d)** TC and Delay on HZ

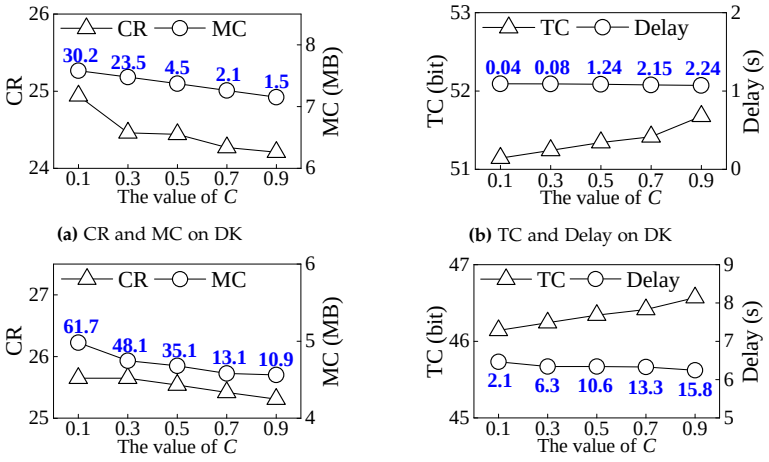
Figure B.7: Effect of k.

prediction and does not perform any offline training.

Performance Metrics. For compression, we use the compression ratio (CR), time delay (Delay), maximum memory cost (MC), and transmission cost (TC) as the performance metrics. For query processing, we use the query time (Time) and transmission cost (TC) as the performance metrics. Specifically, the maximum memory cost records the maximum storage of auxiliary structures (such as hash tables) created for compression over all timestamps. The auxiliary structures are stored in main memory at the edge server, while the compressed data is transmitted to and stored in the centralized cloud. Moreover, both the time delay and transmission cost are reported as average values to process an arriving mapped GPS point at a timestamp. As the experiments are simulated in a vehicular edge computing architecture, the transmission cost is the size of contents to be transferred from the edge server to the centralized cloud. The contents include compressed trajectories and indexes, thus we use the transmission cost as a performance metric of both compression and query processing.

7.2 Experimental Results

Comparison and Scalability. Figure B.6 reports experimental findings when varying the dataset size from 20% to 100%. We use 80%, 10%, and 10% of each dataset for training, validation, and testing for OCT-LSTM, respectively. Thus, all methods are applied to 10% of each dataset in this set of experiments. Since CLEAN is an offline method and takes 192.1 hours to compress 20% Syn, we only report its compression ratio and maximum memory cost on HZ and 20% Syn. First, TRACE outperforms all the baselines in terms of compression ratio



(a) CR and MC on DK

(b) TC and Delay on DK

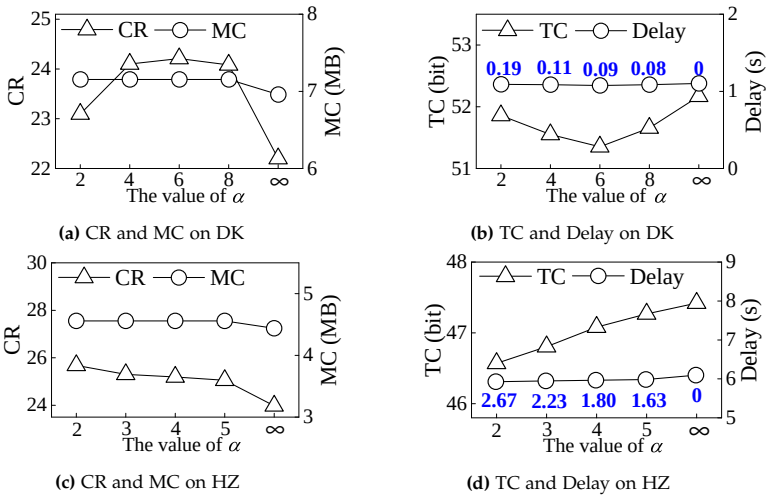
(c) CR and MC on HZ

(d) TC and Delay on HZ

Figure B.8: Effect of C.

and transmission cost. This is mainly due to TRACE's separate compression of paths, speeds, and timestamps that eliminates more redundancy. Second, the compression ratios of TRACE and OCT-LSTM increase slightly, as more subsequences can be referentially compressed and more training data is used. The transmission cost of OCT-LSTM is two orders of magnitude higher than that of TRACE. The reason is that OCT-LSTM needs to update and transmit the model to adapt to new speed patterns. Next, the maximum memory cost and time delay of TRACE exceed those of OCT-LSTM and OCT. This is in line with TRACE's goal of achieving high compression ratios without losing mapped GPS points. Hence, TRACE needs to maintain references in main memory and needs to wait for k mapped GPS points to initialize a factor. In contrast, OCT-LSTM and OCT trade the accuracy for compression efficiency and memory cost, by discarding data that can be predicted within an error bound. However, the maximum memory cost of TRACE never exceeds 8 MB, which we expect is easily accommodated by the edge server layer [29]. The maximum memory cost and the processing time of CLEAN are the highest among the four methods. However, CLEAN is the best among the baselines in terms of compression ratio; thus, it is a good option if the compression can take place offline and the dataset is relatively small. Finally, the time delay of TRACE is roughly independent of the dataset size, while that of OCT-LSTM drops slightly with more data as fewer models are re-trained. The time delay of TRACE on both datasets never exceeds its corresponding default sample interval, i.e., 20s, which suggests that TRACE supports real-time compression.

Effect of k. Figure B.7 reports the results when varying k, the length of k-mers. First, the compression ratios on both DK and HZ first increase and then drop. On one hand, a larger k results in more references due to the



(a) CR and MC on DK

(b) TC and Delay on DK

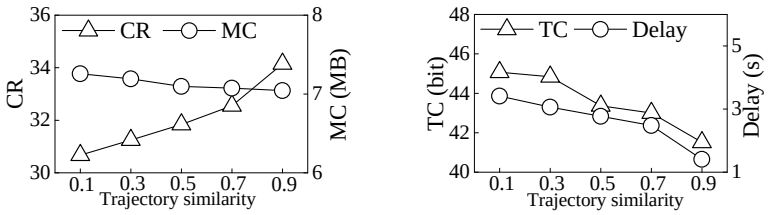
(c) CR and MC on HZ

(d) TC and Delay on HZ

Figure B.9: Effect of α.

higher probability of mismatching among longer sequences, which reduces the compression ratio. On the other hand, a longer subsequence tends to be encoded into one factor, increasing the compression ratio. Second, the maximum memory cost increases with k, because more references lead to more k-mers being stored. Next, the transmission cost has an opposite trend to the compression ratio. The higher the compression ratio we can achieve, the less data is delivered to the centralized cloud. Finally, Figures B.7b and B.7d show that the time delay increases with k in most cases, as the time to initialize each factor rises.

Effect of C. Figure B.8 studies the effect of the deletion coefficient C that controls the amount of outdated data. Specifically, the blue numbers along with the maximum memory cost in Figures B.8a and B.8c report $\frac{km_{max}}{km}$%, where km_{max} is the maximum number of the k-mers in memory over all timestamps using reference deletion, while km is the number without using reference deletion. The figures show that reference deletion function reduces the memory cost substantially. We also report the average processing time (ms) of the reference deletion at each timestamp in Figures B.8b and B.8d (the blue numbers along with the time delay). As observed, the processing time is 2–4 orders of magnitude less than the time delay. In addition, both the compression ratio and the time delay drop with the increasing of C on both DK and HZ. This is because a larger C results in more deletions, which reduces the number of k-mers stored in memory. In this case, an upcoming sequence is more likely to be assigned as a reference. Since the time delay is mainly caused by compressing non-references, it drops with the decrease of non-references. However, the drops are smooth because we only delete k-mers that are unlikely to be referenced.



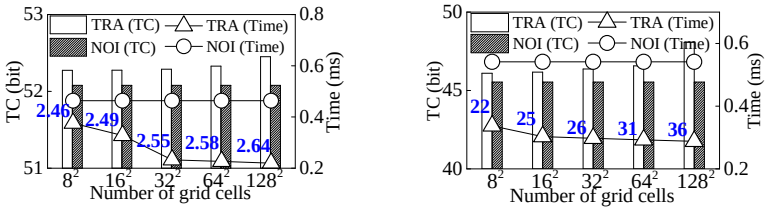
(a) CR and MC **(b)** TC and Delay

Figure B.10: Effect of trajectory similarity.

Effect of α. Figure B.9 reports the impact of the rewriting coefficient α on the compression performance, where α controls the frequency of rewriting and "∞" indicates no rewriting. We see that the compression ratio of TRACE first increases and then drops with the increase of α on DK, while it continues to increase with α on HZ. The reason is that subsequences may be identified wrongly as being frequent if a small α is used, while many truly frequent subsequences may be missed if a very large α is used. Intuitively, the optimal α value highly depends on the dataset. Second, the maximum memory cost of TRACE drops when $\alpha = \infty$, because we do not maintain any auxiliary structures for rewriting in this case. Next, the time delay of TRACE follows the opposite trend of the compression ratio on both datasets, as it is mainly caused by initializing factors. Finally, the blue numbers along with the time delay in Figures B.9b and B.9d are the average processing time (ms) of reference rewriting at each timestamp, which are negligible compared with the time delay.

Effect of Trajectory Similarity. We perform TRACE on Syn0.1, Syn0.3, Syn0.5, Syn0.7, and Syn0.9, denoted as 0.1, 0.3, 0.5, 0.7, and 0.9, respectively, to study the impact of trajectory similarities on compression. Figure B.10 shows that the compression ratio increases and that the transmission cost drops as trajectories become more similar. Moreover, the maximum memory cost and the time delay decrease with the increase of similarity because fewer k-mers are stored and fewer factors are generated. Overall, TRACE achieves higher compression performance on datasets with larger similarity, due to its referential compression.

Effect of *gc*. Figure B.11 reports the transmission cost and query time when varying the number of grid cells, gc. "TRA" denotes our TRACE framework while "NOI" denotes the case of no indexing. It is clear that a larger gc results in higher query efficiency and larger transmission cost. The blue numbers along the TRACE query time line denote the index creation time (μs), which is the average time used on creating/updating indexes for all the arriving locations per timestamp. The index creation time increases slightly with a finer grid granularity, i.e., larger gc. This is because TRACE updates the grid information of streaming trajectories once this changes and then delivers the new information to the centralized cloud to improve the



(a) TC and query time on DK

(b) TC and query time on HZ

Figure B.11: Effect of number of grid cells *gc*.

query efficiency. However, the grid information contributes at most 6.5% to the total transmission cost and the index creation time is negligible compared with the time delay.

8 Related Work

8.1 Raw Data Compression

Raw trajectory compression aims to compact trajectories that have not been map-matched and targets either offline [6, 26, 45] or online settings [8, 18, 27]. REST [45] is the first offline reference-based raw trajectory compression framework. Targeting raw trajectories, it differs very substantially from TRACE. REST compresses the timestamps of a non-reference w.r.t. to that of a reference only when their spatial information are matchable, while TRACE referentially compresses different parts of trajectories separately, which enables the removal of more redundancy. SQUISH [27] is a representative work that aims at reducing data loss and preserves speed information at high accuracy during compression. Deng et al. [8] consider direction-preserving compression in streaming settings and propose an advanced online DPTS algorithm that achieves high compression ratios. Li et al. [18] take into account the special needs of real-time surveillance applications and increase the loading speed of trajectory data very noticeably. Comprehensive experimental evaluations of raw trajectory compression are available [14, 28, 43].

8.2 Network-constrained Compression

Network-constrained trajectory compression leverages the underlying road network to improve compression, which also occurs either offline or online.

Offline Mode. Krogh et al. [17] compress a trajectory by storing only the first and last edges of each shortest path in a trajectory. Ji et al. [13] encode outgoing road segments clockwise based on a pre-computed clockwise code table. Sun et al. [11, 33] propose a two-stage spatial compression algorithm using shortest path and frequent subtrajectory compression. Yang et al. [41]

present a very compact representation that separates the distance information from timestamps. Koide et al. [16] develop a compression technique for spatial information of trajectories and support the retrieval of subpaths. Sui et al. [34] assign each GPS point to the middle point of a segment and propose a road-network partitioning strategy on which the compression ratio depends. CLEAN [44] encodes trajectories by means of frequent patterns. In particular, CLEAN is the first study to perform temporal compression on top of spatial compression and presents novel pattern concatenation and generation techniques that always expand the pattern with the highest support. However, CLEAN needs to count the support of each newly generated pattern in the trajectory dataset. This incurs high main memory and running time costs for large datasets, which precludes it from running in an online manner. UTCQ [19] targets compression of uncertain trajectories; in contrast, TRACE aims to compress streaming trajectories in real-time. Specifically, UTCQ uses an improved TED representation and develops an FJD function to measure the similarity between trajectory instances and to select references in batch mode. Instead, TRACE proposes a more compact speed-based representation and adapts k-mer matching for real-time reference selection.

Online Mode. Four studies exist that target online network-constrained trajectory compression. Chen et al. [3, 4] present a solution that compresses the edge sequences in trajectories by retaining only out-edges with remarkable heading changes and also uses frequent paths trained offline to further improve compression. The solution does not consider the compression of temporal information and locations of trajectories. ONTRAC [32] uses a k-order Markov model to learn frequent paths and apply them to compress incoming edges in real-time. This solution discards the mapped GPS points of the original trajectories and only estimates the time of traversing an edge using a trained model. OCT-LSTM [5] trains models to obtain repetitive movement patterns of the time-distance sequences using historical data and only transfers data when predicted values deviate significantly from the actual ones. Then, re-training is performed at the centralized cloud and the updated model is transmitted to the edge server layer, incurring a high transmission cost. Moreover, OCT-LSTM does not mine frequent paths that exist widely and can enhance compression substantially. If no historical data exists, OCT-LSTM uses linear prediction, called OCT. All the above methods compress trajectories by discarding useful information, but TRACE keeps all of them to maintain data usability. Moreover, both OCT-LSTM and ONTRAC employ an offline training phase as the foundation for their online compression, while TRACE is designed to compress streaming trajectories in a fully online fashion, making it more generally usable in practice.

9 Conclusion and Future Work

We propose a new framework for compressing, transmitting, and querying streaming network-constrained trajectories in real-time. We develop a speed-based and a multiple-references based referential representation to represent trajectories concisely. We propose k-mer matching based online reference selection with reference deletion and rewriting. Deletion reduces the storage cost, while rewriting improves the compression ratio. Moreover, we propose a transmission strategy that reduces the transmission cost and ensures that compressed trajectories are decodable. Finally, we enable querying of compressed streaming trajectories and provide indexing and filtering techniques that accelerate real-time query processing. An experimental study using two real-life datasets and one synthetic dataset shows that the proposed TRACE framework outperforms three baselines [5, 44] in terms of compression ratio and transmission cost. In future research, it is of interest to reduce the latency and memory cost of online compression and to deploy TRACE in a distributed cloud setting.

References

[1] T. Anwar, C. Liu, H. L. Vu, M. S. Islam, and T. Sellis, "Capturing the spatiotemporal evolution in road traffic networks," *TKDE*, vol. 30, no. 8, pp. 1426–1439, 2018.

[2] M. Bierlaire, J. Chen, and J. Newman, "A probabilistic map matching method for smartphone GPS data," *TRANSPORT RES C-EMER*, vol. 26, pp. 78–98, 2013.

[3] C. Chen, Y. Ding, Z. Wang, J. Zhao, B. Guo, and D. Zhang, "Vtracer: When online vehicle trajectory compression meets mobile edge computing," *IEEE Syst J*, vol. 14, no. 2, pp. 1635–1646, 2019.

[4] C. Chen, Y. Ding, X. Xie, S. Zhang, Z. Wang, and L. Feng, "Trajcompressor: An online map-matching-based trajectory compression framework leveraging vehicle heading direction and change," *IEEE trans Intell Transp Syst*, vol. 21, no. 5, pp. 2012–2028, 2019.

[5] J. Chen, Z. Xiao, D. Wang, D. Chen, V. Havyarimana, J. Bai, and H. Chen, "Toward opportunistic compression and transmission for private car trajectory data collection," *IEEE Sens. J.*, vol. 19, no. 5, pp. 1925–1935, 2018.

[6] M. Chen, M. Xu, and P. Franti, "A fast $o(n)$ multiresolution polygonal approximation algorithm for GPS trajectory simplification," *TIP*, vol. 21, no. 5, pp. 2770–2785, 2012.

[7] Y. Chen and L. Tu, "Density-based clustering for real-time stream data," in *SIGKDD*, 2007, pp. 133–142.

[8] Z. Deng, W. Han, L. Wang, R. Ranjan, A. Y. Zomaya, and W. Jie, "An efficient online direction-preserving compression approach for trajectory streaming data," *Future Gener Comput Syst*, vol. 68, pp. 150–162, 2017.

[9] S. Deorowicz and S. Grabowski, "Robust relative compression of genomes with random access," *Bioinformatics*, vol. 27, no. 21, pp. 2979–2986, 2011.

[10] A. Dignös, M. H. Böhlen, and J. Gamper, "Overlap interval partition join," in *SIGMOD*, 2014, pp. 1459–1470.

[11] Y. Han, W. Sun, and B. Zheng, "Compress: A comprehensive framework of trajectory compression in road networks," *TODS*, vol. 42, no. 2, p. 11, 2017.

[12] G. Hu, J. Shao, F. Liu, Y. Wang, and H. Shen, "If-matching: Towards accurate map-matching with information fusion," *TKDE*, vol. 29, no. 1, pp. 114–127, 2016.

[13] Y. Ji, Y. Zang, W. Luo, X. Zhou, Y. Ding, and L. M. Ni, "Clockwise compression for trajectory data under road network constraints," in *ICBDA*, 2016, pp. 472–481.

[14] Jonathan Muckell, Jeong-Hyon Hwang, Catherine T. Lawson and SS Rav, "Algorithms for compressing GPS trajectory data: an empirical evaluation," in *SIGSPATIAL*, 2010, pp. 402–405.

[15] Ken CK Lee, Wang-Chien Lee, Baihua Zheng and Yuan Tian, "ROAD: A new spatial object search framework for road networks," *TKDE*, vol. 24, no. 3, p. 547, 2012.

[16] S. Koide, Y. Tadokoro, C. Xiao, and Y. Ishikawa, "CiNCT: Compression and retrieval for massive vehicular trajectories via relative movement labeling," in *ICDE*, 2018, pp. 1097–1108.

[17] B. Krogh, C. S. Jensen, and K. Torp, "Efficient in-memory indexing of network-constrained trajectories," in *SIGSPATIAL*, 2016, pp. 17–26.

[18] L. Li, X. Xia, and Z. Xiong, "A novel online trajectory compression algorithm for real-time trajectory surveillance applications," in *IMCEC*, 2019, pp. 995–999.

[19] T. Li, R. Huang, L. Chen, C. S. Jensen, and T. B. Pedersen, "Compression of uncertain trajectories in road networks," *PVLDB*, vol. 13, no. 7, pp. 1050–1063, 2020.

[20] Y. Li, G. Li, J. Li, and K. Yao, "Skqai: A novel air index for spatial keyword query processing in road networks," *Inf. Sci.*, vol. 430, pp. 17–38, 2018.

[21] Y. Li, D. W. Gao, W. Gao, H. Zhang, and J. Zhou, "Double-mode energy management for multi-energy system via distributed dynamic event-triggered newton-raphson algorithm," *IEEE T Smart Grid.*, vol. 11, no. 6, pp. 5339–5356, 2020.

[22] Y. Li, W. Gao, W. Gao, H. Zhang, and J. Zhou, "A distributed double-newton descent algorithm for cooperative energy management of multiple energy bodies in energy internet," *IEEE T IND INFORM.*, 2020.

[23] Z. Li, L. Chen, and Y. Wang, "G*-tree: An efficient spatial index on road networks," in *ICDE*, 2019, pp. 268–279.

[24] W. Liu, Y. Zheng, S. Chawla, J. Yuan, and X. Xing, "Discovering spatio-temporal causal interactions in traffic data streams," in *SIGKDD*, 2011, pp. 1010–1018.

[25] Y. Liu, H. Peng, L. Wong, and J. Li, "High-speed and high-ratio referential genome compression," *Bioinformatics*, vol. 33, no. 21, pp. 3364–3372, 2017.

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

[27] J. Muckell, J.-H. Hwang, V. Patil, C. T. Lawson, F. Ping, and S. Ravi, "Squish: an online approach for gps trajectory compression," in *COM.Geo*, 2011, pp. 1–8.

[28] J. Muckell, P. W. Olsen, J.-H. Hwang, S. Ravi, and C. T. Lawson, "A framework for efficient and convenient evaluation of trajectory compression algorithms," in *COM.Geo*, 2013, pp. 24–31.

[29] S. Raza, S. Wang, M. Ahmed, and M. R. Anwar, "A survey on vehicular edge computing: architecture, applications, technical issues, and future directions," *IEEE Wirel Commun*, vol. 19, no. 4, pp. 2322–2358, 2019.

[30] S. J. Samet Hanan and A. Houman, "Scalable network distance browsing in spatial databases," in *SIGMOD*, 2008, pp. 43–54.

[31] W. Sebastian and L. Ulf, "Fresco: Referential compression of highly similar sequences," *TCBB*, vol. 10, no. 5, pp. 1275–1288, 2013.

[32] A. Silva, R. Raghavendra, M. Srivatsa, and A. K. Singh, "Prediction-based online trajectory compression," *arXiv preprint arXiv:1601.06316*, 2016.

[33] R. Song, W. Sun, B. Zheng, and Y. Zheng, "Press: A novel framework of trajectory compression in road networks," *PVLDB*, vol. 7, no. 9, pp. 661–672, 2014.

[34] P. Sui and X. Yang, "A privacy-preserving compression storage method for large trajectory data in road network," *J. Grid Comput.*, vol. 16, no. 2, pp. 229–245, 2018.

[35] S. Taguchi, S. Koide, and T. Yoshimura, "Online map matching with route prediction," *IEEE trans Intell Transp Syst*, vol. 20, no. 1, pp. 338–347, 2018.

[36] Tarique Anwar, Chengfei Liu, Hai L. Vu and Christopher Leckie, "Spatial Partitioning of Large Urban Road Networks," in *EDBT*, 2014, pp. 343–354.

[37] Tarique Anwar, Chengfei Liu, Hai L. Vu and Md Saiful Islam, "Tracking the evolution of congestion in dynamic urban road networks," in *CIKM*, 2016, pp. 2323–2328.

[38] J. Teuhola, "A compression method for clustered bit-vectors," *INFORM PROCESS LETT*, vol. 7, no. 6, pp. 308–311, 1978.

[39] S. Wandelt and U. Leser, "Adaptive efficient compression of genomes," *ALGORITHM MOL BIOL*, vol. 7, no. 1, p. 30, 2012.

[40] Y. Xiang, W. Zhou, and M. Guo, "Flexible deterministic packet marking: An ip traceback system to find the real source of attacks," *IEEE Trans Parallel Distrib Syst*, vol. 20, no. 4, pp. 567–580, 2008.

[41] X. Yang, B. Wang, K. Yang, C. Liu, and B. Zheng, "A novel representation and compression for queries on trajectories in road networks," *TKDE*, vol. 30, no. 4, pp. 613–629, 2017.

[42] H. Yuan and G. Li, "Distributed in-memory trajectory similarity search and join on road network," in *ICDE*, 2019, pp. 1262–1273.

[43] D. Zhang, M. Ding, D. Yang, Y. Liu, J. Fan, and H. Shen, "Trajectory simplification: an experimental study and quality analysis," *PVLDB*, vol. 11, no. 9, pp. 934–946, 2018.

[44] P. Zhao, Q. Zhao, C. Zhang, G. Su, Q. Zhang, and W. Rao, "Clean: frequent pattern-based trajectory spatial-temporal compression on road networks," in *MDM*, 2019, pp. 605–610.

[45] 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 *KDD*, 2018, pp. 2797–2806.

Paper C

Evolutionary Clustering of Streaming Trajectories

Tianyi Li, Lu Chen, Christian S. Jensen, Torben Bach Pedersen, Jilin Hu

The paper has been under reviewed in the
VLDB, pp. XXX–XXX, 2022.

The layout has been revised.

Abstract

The widespread deployment of smartphones and location-enabled, networked in-vehicle devices renders it increasingly feasible to collect streaming trajectory data of moving objects. The continuous clustering of such data can enable a variety of real-time services, such as identifying representative paths or common moving trends among objects in real-time. However, little attention has so far been given to the quality of clusters—for example, it is beneficial to smooth short-term fluctuations in clusters to achieve robustness to exceptional data.

We propose the notion of evolutionary clustering of streaming trajectories, abbreviated ECO, that enhances streaming-trajectory clustering quality by means of temporal smoothing that prevents abrupt changes in clusters across successive timestamps. Employing the notions of snapshot and historical trajectory costs, we formalize ECO and then formulate ECO as an optimization problem and prove that ECO can be performed approximately in linear time, thus eliminating the iterative processes employed in previous studies. Further, we propose a minimal-group structure and a seed point shifting strategy to facilitate temporal smoothing. Finally, we present all algorithms underlying ECO along with a set of optimization techniques. Extensive experiments with two real-life datasets offer insight into ECO and show that it outperforms state-of-the-art solutions in terms of both clustering quality and efficiency.

1 Introduction

It is increasingly possible to equip moving objects with positioning devices that are capable of transmitting object positions to a central location in real time. Examples include people with smartphones and vehicles with built-in navigation devices or tracking devices. This scenario opens new opportunities for the real-time discovery of hidden mobility patterns. These patterns allow characterizing individual mobility for a certain time interval and enable a broad range of important services and applications such as route planning [33, 41], intelligent transportation management [34], and road infrastructure optimization [35].

As a typical moving pattern discovery approach, clustering aims to group a set of trajectories into comparatively homogeneous clusters to extract representative paths or movement patterns shared by moving objects. Considering a streaming setting, many works are proposed to cluster the trajectories in real-time [5, 7–9, 16, 21, 22, 32, 39]. However, existing real-time clustering methods focus on the most current data, achieving low computational cost at the expense of clustering quality [36]. In streaming settings, clusterings should be robust to short-term fluctuations in the underlying trajectory data, which may be achieved by means of smoothing [6]. An example illustrates this.

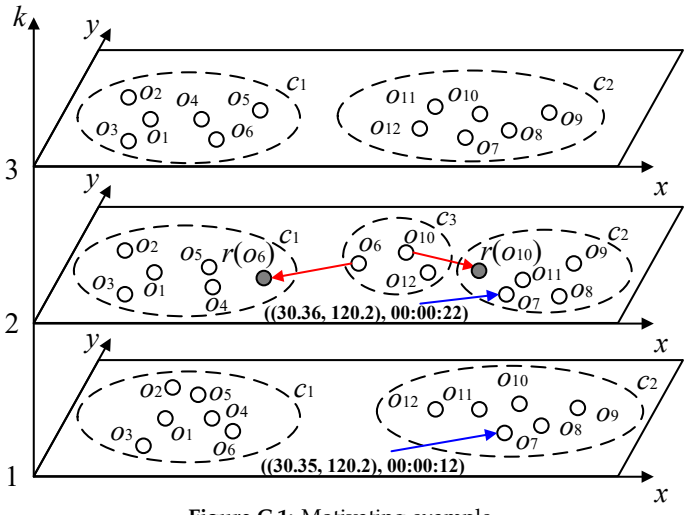


Figure C.1: Motivating example

Example 1.1
Figure C.1 shows the trajectories of 12 moving objects at three timestamps, $k = 1, 2, 3$. Traditional clustering algorithms return the two clusters $c_1 = \{o_1, o_2, o_3, o_4, o_5, o_6\}$ and $c_2 = \{o_7, o_8, o_9, o_{10}, o_{11}, o_{12}\}$ at the first timestamp, the three clusters $c_1 = \{o_1, o_2, o_3, o_4, o_5\}$, $c_2 = \{o_7, o_8, o_9, o_{11}\}$, and $c_3 = \{o_6, o_{10}, o_{12}\}$ at the second timestamp, and the same two clusters at the third timestamp as at the first timestamp.

The underlying reason for this result is the unusual behavior of objects o_6 and o_{10} at the second timestamp. Clearly, returning the same two stable clusters for all three timestamps is a more robust and better-quality result. A naive approach to eliminating the effect of the two objects' unusual behavior is to perform cleaning before clustering. However, studies of on two real-life datasets show that among the trajectories that cause the mutations of clusterings, 88.9% and 75.9% of the trajectories follow the speed constraint, while 97.8% and 96.1% of them are categorized as inliers [2]. Moreover, in real-time applications, it is impractical to correct previous clusterings retroactively. Hence, it is difficult for existing cleaning techniques to facilitate smoothly shifted clustering sequences [15, 18, 28].

However, this problem can be addressed by applying evolutionary clustering [3, 4, 6, 11, 14, 17, 23, 25, 36, 37], where a good current clustering result is one that fits the current data well, while not deviating too much from the recent history of clusterings. Specifically, *temporal smoothness* is integrated into the measure of clustering quality [6]. This way, evolutionary clustering is able to outperform traditional clustering as it can reflect long term trends while being robust to short-term variability. Put differently, applying evolu-

tionary clustering to trajectories can mitigate adverse effects of intermittent noise on clustering and present users with smooth and consistent movement patterns. In Example 1.1, clustering with temporal consistency is obtained if o_6 is smoothed to $r(o_6)$ and o_{10} is smoothed to $r(o_{10})$ at the second timestamp. Motivated by this, we study evolutionary clustering of trajectories.

Existing evolutionary clustering studies target dynamic networks and are not suitable for trajectory applications, mainly for three reasons. First, the solutions are designed specifically for dynamic networks, which differ substantially from two-dimensional trajectory data. Second, the movement in trajectories is generally much faster than the evolution of dynamic networks, which renders the temporal smoothness used in existing studies too "strict" for trajectories. Third, existing studies often optimize the clustering quality iteratively at each timestamp [3, 12, 17, 23, 24, 37], which is computationally costly and is infeasible for large-scale trajectories.

We propose an efficient and effective method for evolutionary clustering of streaming trajectories (**ECO**). First, we adopt the idea of neighbor-based smoothing [17] and develop a structure called *minimal group* that is summarized by a *seed point* in order to facilitate smoothing. Second, following existing studies [3, 12, 23, 24, 36, 37], we formulate ECO as an optimization problem that employs the new notions of snapshot cost and historical cost. The snapshot cost evaluates the true concept shift of clustering defined according to the distances between smoothed and original locations. The historical cost evaluates the temporal distance between locations at adjacent timestamps by the *degree of closeness*. Next, we prove that the proposed optimization function can be decomposed and that each component can be solved approximately in constant time. The effectiveness of smoothing is further improved by a *seed point shifting* strategy. Finally, we introduce a grid index structure and present algorithms for each component of evolutionary clustering along with a set of optimization techniques, to improve clustering performance. The paper's main contributions are summarized as follows,

- We formalize ECO problem. To the best of our knowledge, this is the first proposal for streaming trajectory clustering that takes into account temporal smoothness.
- We formulate ECO as an optimization problem, based on the new notions of snapshot cost and historical cost. We prove that the optimization problem can be solved approximately in linear time.
- We propose a *minimal group* structure to facilitate temporal smoothing and a *seed point shifting* strategy to improve clustering quality of evolutionary clustering. Moreover, we present all algorithms needed to enable evolutionary clustering, along with a set of optimization techniques.
- Extensive experiments on two real-life datasets show that ECO advances the state-of-the-arts in terms of both clustering quality and efficiency.

The rest of paper is organized as follows. We present preliminaries in Section 2. We formulate the problem in Section 3 and derive its solution in Section 4. Section 5 presents the algorithms and optimization techniques. Section 6 covers the experimental study. Section 7 reviews related work, and Section 8 concludes and offers directions for future work.

2 Preliminaries

2.1 Data Model

Definition 2.1
A **GPS record** is a pair (l, t), where t is a **timestamp** and $l = (x, y)$ is the **location**, with x being a longitude and y being a latitude.

Definition 2.2
A **streaming trajectory** o is an unbounded ordered sequence of GPS records, $\langle (o.l_1, o.t_1), (o.l_2, o.t_2) \cdots \rangle$.

The GPS records of a trajectory may be transmitted to a central location in an unsynchronized manner. To avoid this affecting the subsequent processing, we adopt an existing approach [5] and discretize time into short intervals that are indexed by integers. We then map the timestamp of each GPS record to the index of the interval that the timestamp belongs to. In particular, we assume that the start time is 00:00:00 UTC, and we partition time into intervals of duration $\Delta t = 10s$. Then time series ⟨00:00:01, 00:00:12, 00:00:20, 00:00:31, 00:00:44⟩ and ⟨00:00:00, 00:00:13, 00:00:21, 00:00:31, 00:00:40⟩ are both mapped $\langle 0, 1, 2, 3, 4 \rangle$. We call such a sequence a discretized time sequence and call each discretized timestamp a ***time step*** dt. We use trajectory and streaming trajectory interchangeably.

Definition 2.3
A trajectory is **active** at time step $dt = [t_1, t_2]$ if it contains a GPS record (l, t) such that $t \in [t_1, t_2]$.

Definition 2.4
A **snapshot** $\mathcal{O}_k$ is the set of trajectories that are active at time step dt_k.

Figure C.1 shows three snapshots $\mathcal{O}_1$, $\mathcal{O}_2$, and $\mathcal{O}_3$, each of which contains twelve trajectories. Given the start time 00:00:00 and $\Delta t = 10$, $(o_7.l, o_7.t)$ arrives at dt_1 because 00:00:12 is mapped to 1. For simplicity, we use o in figures to denote $o.l$. The interval duration Δt is the default sample interval of the dataset. Since deviations between the default sample interval and the actual intervals are small [20], we can assume that each trajectory o has at most one GPS record at each time step dt_k. If this is not the case for a trajectory o, we simply keep o's

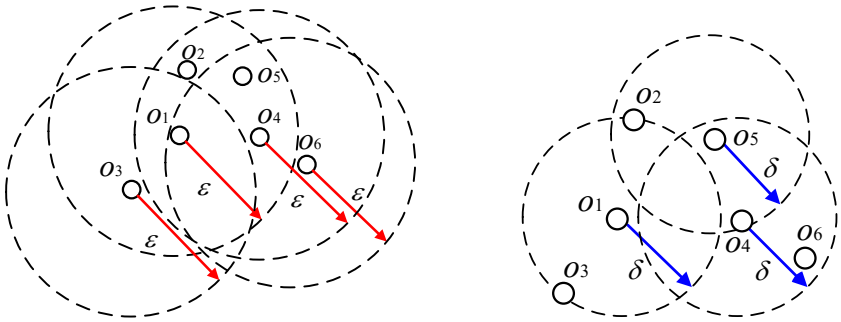
Table C.1: Frequently used notation

Notation	Description
o	A trajectory
dt_k	The k^{th} time step
$o.l_k$, $o.t_k$	The location and timestamp of o at dt_k
$o.l$, $o.t$	A simplification of $o.l_k$, $o.t_k$ at dt_k
$o.\tilde{l}$, $o.\tilde{t}$	A simplification of $o.l_{k-1}$, $o.t_{k-1}$ at dt_k
$\mathcal{O}_k$	A set of trajectories at dt_k
$r(o)$	An adjustment of $o.l$
$\mathcal{R}_k$	The set of adjustments of $\mathcal{O}_k$
s	A seed point of o at the current time step dt_k
$\tilde{s}$	A seed point of o at the previous time step dt_{k-1}
$\mathcal{S}_k$	The set of seed points at dt_k
$\mathcal{M}_k(s)$	A minimal group summarized by a seed point s at dt_k
$\mathcal{SC}_k(r(o))$	The snapshot cost of a trajectory o w.r.t. $r(o)$ at dt_k
$\mathcal{TC}_k(r(o))$	The historical cost of a trajectory o w.r.t. $r(o)$ at dt_k
c	A cluster c
$\mathcal{C}_k$	The set of clusters obtained at dt_k

earliest GPS at the time step. This simplifies the subsequent clustering. Thus, the GPS record of o at dt_k is denoted as $(o.l_k, o.t_k)$. If a trajectory o is active at both dt_{k-1} and dt_k and the current time step is dt_k, $o.l_k$ and $o.t_k$ are simplified as $o.l$ and $o.t$, and $o.l_{k-1}$ and $o.t_{k-1}$ are simplified as $o.\tilde{l}$ and $o.\tilde{t}$. At time step dt_2 ($k = 2$) in Figure C.1, $o_7.\tilde{l} = o_7.l_1 = (30.35, 120.2)$, $o_7.\tilde{t} = o_7.t_1 =$ 00:00:12, $o_7.l = o_7.l_2 = (30.36, 120.2)$, and $o_7.t = o_7.t_2 =$ 00:00:22.

Definition 2.5

A **θ-neighbor set** of a streaming trajectory $o\,(\in \mathcal{O}_k)$ at the time step dt_k is $\mathcal{N}_\theta(o) = \{o' | o' \in \mathcal{O}_k \wedge d(o.l, o'.l) \leq \theta\}$,where $d(\cdot)$ is Euclidean distance and θ is a distance threshold. $|\mathcal{N}_\theta(o)|$ is called the **local density** of o w.r.t. θ at dt_k.



(a) Core points $o_i\,(1 \leq i \leq 6 \wedge i \neq 3)$ ($minPts = 3$) **(b)** Seed points o_1 and o_4 ($\rho = 3$)

Figure C.2: $o_i\,(1 \leq i \leq 6)$ at dt_1 in Figure C.1

Figure C.2 plots $o_i\,(1 \leq 1 \leq 6)$ at dt_1 from Figure C.1, where $\mathcal{N}_\delta(o_1) = \{o_1, o_2, o_3\}$.

2.2 DBSCAN

We adopt a well-known density-based clustering approach, DBSCAN [10], for clustering. DBSCAN relies on two parameters to characterize density or sparsity, i.e., positive values ε and *minPts*.

Definition 2.6
A trajectory $o \in \mathcal{O}_k$ is a **core point** w.r.t. ε and *minPts*, if $\mathcal{N}_\varepsilon(o) \geq minPts$.

Definition 2.7
A trajectory $o \in \mathcal{O}_k$ is **density reachable** from another trajectory $o' \in \mathcal{O}_k$, if a sequence of trajectories $o_1, o_2, \cdots o_n\ (n \geq 2)$ exists such that (i) $o_1 = o'$ and $o_n = o$; (ii) $o_w\ (1 \leq w < n)$ are core points; and (iii) $d(o_w, o_{w+1}) \leq \varepsilon\ (1 \leq w < n)$.

Definition 2.8
A trajectory $o \in \mathcal{O}_k$ is **connected** to another trajectory o' if a trajectory o'' exists such that both o and o' are density reachable from o''.

Definition 2.9
A non-empty subset of trajectories of $\mathcal{O}_k$ is called a **cluster** c, if c satisfies the following conditions:

- Connectivity: $\forall o, o' \in c$, o is connected to o';
- Maximality: $\forall o, o' \in \mathcal{O}_k$, if $o \in c$ and o' is density reachable from o, then $o' \in c$.

Definition 2.9 indicates that a cluster is formed by a set of core points and their density reachable points. Given ε and *minPts*, $o \in \mathcal{O}_k$ is an ***outlier***, if it is not in any cluster; $o \in \mathcal{O}_k$ is a ***border point***, if $\mathcal{N}_\varepsilon(o) < minPts$ and $d(o, o') \leq \epsilon$, where o' is a core point.

Definition 2.10
A **clustering result** $\mathcal{C}_k = \{c_1, c_2, \cdots, c_n\}$ is a set of clusters obtained from the snapshot $\mathcal{O}_k$.

Example 2.1
In Figure C.1, $\mathcal{C}_1$ has two clusters $c_1 = \{o_1, o_2, o_3, o_4, o_5,$
$o_6\}$ and $c_2 = \{o_7, o_8, o_9, o_{10}, o_{11}, o_{12}\}$. Further, $o_i\,(1 \leq i \leq 6 \wedge i \neq 3)$ in Figure C.2a are core points.

2.3 Evolutionary Clustering

Evolutionary clustering is the problem producing a sequence of clusterings from streaming data; that is, clustering for each snapshot. It takes into account the smoothness characteristics of streaming data to obtain high-quality clusterings [3]. Specifically, two quality aspects are considered:

- High historical quality: clustering $\mathcal{C}_k$ should be similar to the previous clustering $\mathcal{C}_{k-1}$;
- High snapshot quality: $\mathcal{C}_k$ should reflect the true concept shift of clustering, i.e., remain faithful to the data at each time step.

Evolutionary clustering uses a cost function $\mathcal{F}_k$ that enables trade-offs between historical quality and snapshot quality at each time step dt_k [3],

$$\mathcal{F}_k = \mathcal{SC}_k(\mathcal{C}_o, \mathcal{C}_k) + \alpha \cdot \mathcal{TC}_k(\mathcal{C}_{k-1}, \mathcal{C}_k) \tag{C.1}$$

$\mathcal{F}_k$ is the sum of two terms: a snapshot cost ($\mathcal{SC}_k$) and a historical cost ($\mathcal{TC}_k$). The snapshot cost $\mathcal{SC}_k$ captures the similarity between clustering $\mathcal{C}_k$ and clustering $\mathcal{C}_o$ that is obtained without smoothing. The smaller $\mathcal{SC}_k$ is, the better the snapshot quality is. The historical cost $\mathcal{TC}_k$ measures how similar clustering $\mathcal{C}_k$ and the previous clustering $\mathcal{C}_{k-1}$ are. The smaller $\mathcal{TC}_k$ is, the better the historical quality is. Parameter $\alpha\,(> 0)$ enables controlling the trades-off between snapshot quality and historical quality.

3 Problem Statement

We start by presenting two observations, based on which, we define the problem of evolutionary clustering of streaming trajectories.

3.1 Observations

Gradual evolutions of travel companions As pointed out in a previous study [32], movement trajectories represent continuous and gradual location changes, rather than abrupt changes, implying that co-movements among trajectories also change only gradually over time. Co-movement may be caused by (i) physical constraints of both road networks and vehicles, and (ii) vehicles may have close relationships, e.g., they may belong to the same fleet or may target the same general destination [32].

Uncertainty of "border" points Even with the observation that movements captured by trajectories are not dramatic during a short time, border points are relatively more likely to leave their current cluster at the next time step

than core points. This is validated by statistics from two real-life datasets. Specifically, among the trajectories shifting to another cluster or becoming an outlier during the next time steps, 75.0% and 61.5% are border points in the two real-life datasets.

3.2 Problem Definition

Cost embedding Existing evolutionary clustering studies generally perform temporal smoothing on the clustering result [3, 6, 12, 37]. Specifically, they adjust $\mathcal{C}_k$ iteratively so as to minimize Formula C.1, which incurs very high cost. We adopt cost embedding [17], which pushes down the cost formula from the clustering result level to the data level, thus enabling flexible and efficient temporal smoothing. However, the existing cost embedding technique [17] targets dynamic networks only. To apply cost embedding to trajectories, we propose a minimal group structure and snapshot and historical cost functions.

Snapshot cost $\mathcal{SC}_k$ We first define the notion of an "adjustment" of a trajectory.

Definition 3.1
An **adjustment** $r_k(o)$ is a location of a trajectory o obtained through smoothing at dt_k. Here, $r_k(o) \neq r_k(o')$ if $o \neq o'$. The set of adjustments in $\mathcal{O}_k$ is denoted as $\mathcal{R}_k$.

We simplify $r_k(o)$ to $r(o)$ if the context is clear. In Figure C.1, $r(o_6)$ is an adjustment of o_6 at dt_2. According to Formula C.1, the snapshot cost measures how similar the current clustering result $\mathcal{C}_k$ is to the original clustering result $\mathcal{C}_o$. Since we adopt cost embedding that smooths trajectories at the data level, the snapshot cost of a trajectory o w.r.t. its adjustment $r(o)$ at dt_k (denoted as $\mathcal{SC}_k(r(o))$) is formulated as the deviation between o and $r(o)$ at dt_k:

$$\mathcal{SC}_k(r(o)) = d(r(o), o.l)^2 \quad s.t. \quad d(r(o), o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t}), \tag{C.2}$$

where μ is a speed constraint of the road network. Formula C.2 requires that any adjustment $r(o)$ must follow the speed constraint. Obviously, the larger the distance between $o.l$ and its adjustment $r(o)$, the higher the snapshot cost.

Historical cost $\mathcal{TC}_k$ As discussed in Section 2.3, one of the goals of evolutionary clustering is smoothing the change of clustering results during adjacent time steps. Since we push down the smoothing from the cluster level to trajectory level, the problem becomes one of ensuring that each trajectory represents a smooth movement. According to the first observation in Section 3.1, gradual location changes lead to stable co-movement relationships

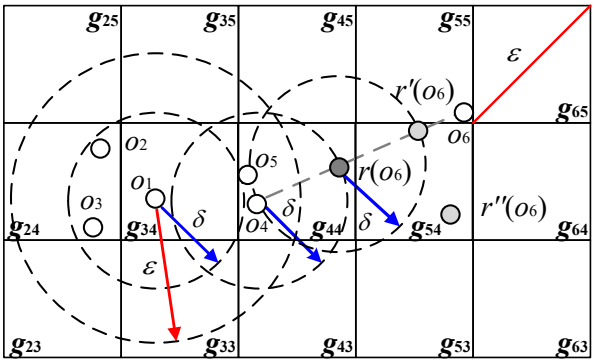


Figure C.3: o_i $(1 \leq i \leq 6)$ at dt_2 in Figure C.1 $(\rho = 3)$

among trajectories during short periods of time. Thus, similar to neighbor-based smoothing in dynamic communities [17], it is reasonable to smooth the location of each trajectory in the current time step using its neighbours at the previous time step. However, the previous study [17] smooths the distance between each pair of neighboring nodes. Simply applying this to trajectories may degrade the performance of smoothing if a "border" point is involved. Recall the second observation of Section 3.1 and assume that $o_1.l$ is smoothed according to $o_3.l$ at dt_2 in Figures C.1 and C.2 . As o_3 is a border point at dt_1 with a higher probability to leave the cluster c_1 at dt_2, using o_3 to smooth o_1 may result in o_1 also leaving c_1 or being located at the border of c_1 at dt_2. The first case may incur an abrupt change to the clustering while the second case may degrade the intra-density of c_1 $(\in \mathcal{C}_2)$ and increase the inter-density of clusters in $\mathcal{C}_2$. To tackle this problem, we model neighboring trajectories as minimal groups summarized by seed points.

Definition 3.2
A **seed point** $s\,(s \in \mathcal{S}_k)$ summarizes a **minimal group** $\mathcal{M}_k(s) = \{o \in \mathcal{O}_k | \, d(o,s) \leq \delta \wedge \forall s' \in \mathcal{S}_k\,(s' \neq s \Rightarrow d(o,s) \leq d(o,s'))\}$ at dt_k, where $\delta\,(0 < \delta \leq \varepsilon)$ is a given parameter, and $\mathcal{S}_k\,(\mathcal{S}_k \subset \mathcal{O}_k)$ is a seed point set at dt_k. The cardinality of $\mathcal{M}_k(s)$, $|\mathcal{M}_k(s)|$, exceeds a parameter ρ. Any trajectory o in $\mathcal{M}_k(s)$ that is different from s is called a **non-seed point**. Note that, $\mathcal{M}_k(s) \cap \mathcal{M}_k(s') = \emptyset$ if $s \neq s'$.

Given the current time step dt_k, we use s to denote the seed point of o at dt_k (i.e., $o \in \mathcal{M}_k(s)$), while use $\tilde{s}$ to denote that at dt_{k-1} (i.e., $o \in \mathcal{M}_{k-1}(\tilde{s})$).

Example 3.1
In Figure C.2b, there are two minimal groups, i.e., $\mathcal{M}_1(o_1) = \{o_1, o_2, o_3\}$ and $\mathcal{M}_1(o_4) = \{o_4, o_5, o_6\}$. In Figure C.3, there is only one minimal group before smoothing, i.e., $\mathcal{M}_2(o_1) = \{o_1, o_2, o_3\}$. Further, given the current $k = 2$, both s and $\tilde{s}$ of o_2 is o_1 and $\tilde{s}.\tilde{l} = o_1.l_1$.

We propose to use the location of a seed point s to smooth the location of a non-seed point o ($o \in \mathcal{M}_{k-1}(\tilde{s})$) at dt_k. In order to guarantee the effectiveness of smoothing, Definition 3.2 gives two constraints when generating minimal groups: (i) $d(o,s) < \delta$ ($\delta \leq \varepsilon$) and (ii) $|\mathcal{M}_k(s)| \geq \rho$. Setting δ to a small value, the first constraint ensures that $o \in \mathcal{M}_k(s)$ are close neighbors at dt_k. Specifically, we require $\delta \leq \varepsilon$ because this makes it very likely that trajectories in the same minimal group are in the same cluster. The second constraint avoids small neighbor sets $\mathcal{N}_\delta(s)$. Specifically, using an "uncertain border" point as a "pivot" to smooth the movement of other trajectories may lead to an abrupt change between clusterings or a low-quality clustering (according to the quality metrics of traditional clustering). We present the algorithm for generating minimal groups in Section 5.2.

Based on the above analysis, we formalize the historical cost of o w.r.t. its adjustment $r(o)$ at dt_k, denoted as $\mathcal{TC}_k(r(o))$, as follows.

$$\begin{aligned} &\mathcal{TC}_k(r(o)) = \left(\lceil \frac{d(r(o), \tilde{s}.l)}{\delta} \rceil - 1 \right)^2 \\ &s.t.\, d(r(o), o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t}), \end{aligned} \tag{C.3}$$

where $o \in \mathcal{M}_{k-1}(\tilde{s}) \backslash \{\tilde{s}\}$. Given the threshold δ, the larger the distance between $r(o)$ and $\tilde{s}.l$, the higher the historical cost. Here, we use the degree of closeness (i.e., $\lceil \frac{d(r(o),\tilde{s}.l)}{\delta} \rceil - 1$) instead of $d(r(o), \tilde{s}.l)$ to evaluate the historical cost, due to two reasons. First, constraining the exact relative distance between any two trajectories during a time interval may be too restrictive, as it varies over time in most cases. Second, using the degree of closeness to constrain the historical cost is sufficient to obtain a smooth evolution of clusterings.

Total cost $\mathcal{F}_k$ Formulas C.2 and C.3 give the snapshot cost and historical cost for each trajectory o w.r.t. its adjustment $r(o)$, respectively. However, the first measures the distance while the latter evaluates the degree of proximity. Thus, we normalize them to a specific range $[0, 1)$:

$$\mathcal{SC}_k(r(o)) = \left(\frac{d(r(o), o.l)}{4\mu \cdot \Delta t + \delta} \right)^2 s.t.\, d(r(o), o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t}) \tag{C.4}$$

$$\begin{aligned} &\mathcal{TC}_k(r(o)) = \left(\frac{\lceil \frac{d(r(o),\tilde{s}.l)}{\delta} \rceil - 1}{\frac{4\mu \cdot \Delta t + \delta}{\delta}} \right)^2 \\ &s.t.\, d(r(o), o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t}), \end{aligned} \tag{C.5}$$

where $o \in \mathcal{M}_{k-1}(\tilde{s}) \backslash \{\tilde{s}\}$ and Δt is the duration of a time step. Clearly, $\mathcal{SC}_k(r(o)) \geq 0$ and $\mathcal{TC}_k(r(o)) \geq 0$. Thus, we only need to prove $\mathcal{SC}_k(r(o)) < 1$ and $\mathcal{TC}_k(r(o)) < 1$.

Lemma 3.1
If $d(o.l, o.\tilde{l}) \leq (o.t - o.\tilde{t}) \cdot \mu$ then $d(r(o), o.l) \leq 4\mu \cdot \Delta t$.

Proof. According to our strategy of mapping original timestamps (Section 2.1), $o.t - o.\tilde{t} \leq 2\Delta t$. Considering the speed constraint of the road network, $d(r(o), o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t})$. Further, $d(r(o), o.l) \leq d(o.l, o.\tilde{l}) + d(r(o), o.\tilde{l})$ due to the triangle inequality. Thus, we get $d(r(o), o.l) \leq d(o.l, o.\tilde{l}) + 2\mu \cdot \Delta t$. Since $d(o.l, o.\tilde{l}) \leq (o.t - o.\tilde{t}) \cdot \mu$, $d(r(o), o.l) \leq 4\mu \cdot \Delta t$. □

It follows from Lemma 3.1 that $\mathcal{SC}_k(r(o)) < 1$ if $d(o.l, o.\tilde{l}) \leq (o.t - o.\tilde{t}) \cdot \mu$. However, $d(o.l, o.\tilde{l}) \leq (o.t - o.\tilde{t}) \cdot \mu$ does not necessarily hold. To address this problem, we pre-process $o.l$ according to $o.\tilde{l}$ so that it follows the speed constraint before conducting evolutionary clustering. The details are given in Section 4.3.

Lemma 3.2
If $d(\tilde{s}.l, \tilde{s}.\tilde{l}) \leq (\tilde{s}.t - \tilde{s}.\tilde{t}) \cdot \mu$ then $d(r(o), \tilde{s}.l) \leq 4\mu \cdot \Delta t + \delta$.

Proof. We have $d(r(o), o.\tilde{l}) \leq 2\mu \cdot \Delta t$ according to Lemma 3.1. Further, $d(o.\tilde{l}, \tilde{s}.\tilde{l}) \leq \delta$ according to Definition 3.2. Since $d(r(o), \tilde{s}.l) \leq d(r(o), o.\tilde{l}) + d(o.\tilde{l}, \tilde{s}.l) \leq d(r(o), o.\tilde{l}) + d(o.\tilde{l}, \tilde{s}.\tilde{l}) + d(\tilde{s}.\tilde{l}, \tilde{s}.l)$, we get $d(r(o), \tilde{s}.l) \leq 4\mu \cdot \Delta t + \delta$. □

According to Lemma 3.2, we can derive $\lceil \frac{4\mu \cdot \Delta t + \delta}{\delta} \rceil - 1 < \frac{4\mu \cdot \Delta t + \delta}{\delta}$ and thus $\mathcal{TC}_k(r(o)) < 1$. Letting $4\mu \cdot \Delta t + \delta = \pi$, the total cost $\mathcal{F}_k$ is:

$$\mathcal{F}_k = \sum_{o, \tilde{s} \in \Theta_k \wedge o \neq \tilde{s}} \frac{1}{\pi^2} \left(d(r(o), o.l)^2 + \alpha \cdot \left(\delta \cdot \left(\lceil \frac{d(r(o), \tilde{s}.l)}{\delta} \rceil - 1 \right) \right)^2 \right) \quad \text{(C.6)}$$
$$s.t. \forall o \in \Theta_k \, (d(r(o), o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t})),$$

where $\Theta_k = \mathcal{O}_k \cap (\bigcup_{\tilde{s} \in \mathcal{S}_{k-1}} \mathcal{M}_{k-1}(\tilde{s}))$. Formula C.6 indicates that we do not smooth the location of o at dt_k if o is not summarized in any minimal group at dt_{k-1}. This is in accordance with the basic idea that we conduct smoothing by exploring the neighboring trajectories. We can now formulate our problem.

Definition 3.3
Given a snapshot $\mathcal{O}_k$, a set of previous minimal groups $\bigcup_{\tilde{s} \in \mathcal{S}_{k-1}} \mathcal{M}_{k-1}(\tilde{s})$, a time duration Δt, a speed constraint μ, and parameters α, δ, ε, *minPts*, and ρ, **evolutionary clustering of streaming trajectories** (**ECO**) is to

- find a set of adjustments $\mathcal{R}_{k_{opt}}$, such that $\mathcal{R}_{k_{opt}} = \arg\min_{\mathcal{R}_k} \mathcal{F}_k$;
- compute a set of clusters $\mathcal{C}_k$ over $\mathcal{R}_{k_{opt}}$.

Specifically, each adjustment of $o.l \in \mathcal{R}_{k_{opt}}$ is denoted as $r_{opt}(o)$ and is then used as the previous location of o (i.e. $o.\tilde{l}$) at dt_{k+1} for evolutionary clustering.

Example 3.2
Following Example 3.1, ECO first finds a set of adjustments $\mathcal{R}_{2_{opt}} = \{r_{opt}(o_i)|1 \leq i \leq 12\}$ at dt_2. Then, it performs clustering over $\mathcal{R}_{2_{opt}}$ and gets $\mathcal{C}_2 = \{c_1, c_2\}$, where $c_1 = \{o_i|1 \leq i \leq 6\}$ and $c_2 = \{o_i|7 \leq i \leq 12\}$. Note that we only show $r_{opt}(o_6)(= r(o_6))$ and $r_{opt}(o_{10})(= r(o_{10}))$ in Figures C.1 and C.3 because $r_{opt}(o_i) = o_i.l\ (1 \leq i \leq 12 \wedge i \neq 6 \wedge i \neq 10)$ at dt_2.

Clearly, the objective function in Formula C.6 is neither continuous nor differentiable. Thus, computing the optimal adjustments using existing solvers involves iterative processes [30] that are too expensive for online scenarios. We thus prove that Formula C.6 can be solved approximately in linear time in Section 4.

4 Computation of Adjustments

Given the current time step dt_k, we start by decomposing $\mathcal{F}_k$ at the unit of minimal groups as follows,

$$\begin{aligned}\mathcal{F}_k &= \sum_{\tilde{s}\in\mathcal{S}_{k-1}} f_k(\tilde{s}.l) \\ &= \sum_{\tilde{s}\in\mathcal{S}_{k-1}} \sum_{o\in\Omega} \left(d(r(o), o.l)^2 + \alpha \cdot \left(\delta \cdot \left(\lceil \frac{d(r(o), \tilde{s}.l)}{\delta} \rceil - 1 \right) \right)^2 \right) \\ &s.t. \forall o \in \Theta_k \left(d(r(o), o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t}) \right),\end{aligned} \tag{C.7}$$

where $\Theta_k = \mathcal{O}_k \cap (\bigcup_{\tilde{s}\in\mathcal{S}_{k-1}} \mathcal{M}_{k-1}(\tilde{s}))$, $\Omega = \mathcal{M}_{k-1}(\tilde{s}) \backslash \{\tilde{s}\}$, $r(o)$ is the adjustment of $o.l$ at dt_k, $\tilde{s}$ is the seed point of o at dt_{k-1}, and $\tilde{s}.l$ is the location of $\tilde{s}$ at dt_k. We omit the multiplier $\frac{1}{\pi^2}$ from Formula C.6 because Δt, μ, and δ are constants and do not affect the results.

4.1 Linear Time Solution

We show that Formula C.7 can be solved approximately in linear time. However, Formula C.7 uses each previous seed point $\tilde{s}$ for smoothing, and such points may also exhibit unusual behaviors from dt_{k-1} to dt_k. Moreover, $\tilde{s}$ may not be in $\mathcal{O}_k$. We address these problems in Section 4.2 by proposing a seed point shifting strategy, and we assume here that $\tilde{s} \in \mathcal{O}_k$ has already been smoothed, i.e., $r(\tilde{s}) = \tilde{s}.l$.

Lemma 4.1
$\mathcal{F}_k$ achieves the minimum value if each $f_k(\tilde{s}.l)\ (\tilde{s} \in \mathcal{S}_{k-1})$ achieves the minimum value.

Proof. To prove this, we only need to prove that $f_k(\tilde{s}.l)$ and $f_k(\tilde{s}'.\tilde{l})$ $(\tilde{s} \neq \tilde{s}' \wedge \tilde{s}, \tilde{s}' \in \mathcal{S}_{k-1})$ do not affect each other. This can be established easily, as we require $\mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{M}_{k-1}(\tilde{s}') = \emptyset$. We thus omit the details due to space limitation. □

Lemma 4.1 implies that Formula C.7 can be solved by minimizing each $f_k(\tilde{s}.l)$ $(\tilde{s} \in \mathcal{S}_{k-1})$. Next, we further "push down" the cost shown in Formula C.7 to each pair of o $(o \in \mathcal{M}_{k-1}(\tilde{s}) \backslash \{\tilde{s}\})$ and $\tilde{s}$.

$$\begin{aligned} f_k(r(o), \tilde{s}.l) \quad &= \quad \left(d(r(o), o.l)^2 + \alpha \cdot \left(\delta \cdot \left(\lceil \frac{d(r(o), \tilde{s}.l)}{\delta} \rceil - 1 \right) \right)^2 \right) \\ &s.t. \quad d(r(o), o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t}) \end{aligned} \tag{C.8}$$

Lemma 4.2
$f_k(\tilde{s}.l)$ achieves the minimum value if each $f_k(r(o), \tilde{s}.l)$ $(o \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k \backslash \{\tilde{s}\})$ achieves the minimum value.

Proof. The proof is straightforward, because $f_k(r(o), \tilde{s}.l)$ and $f_k(r'(o), \tilde{s}.l)$ $(o, o' \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k \backslash \{\tilde{s}\} \wedge o \neq o')$ are independent of each other. □

According to Lemma 4.2, the problem is simplified to computing $r_{opt}(o) = \arg\min_{r(o)} f_k(r(o), \tilde{s}.l)$ $(o \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k \backslash \{\tilde{s}\})$ given $\tilde{s}$. However, Formula C.8 is still intractable as its objective function is not continuous. We thus aim to transform it into a continuous function. Before doing so, we cover the case where the computation of $r_{opt}(o)$ w.r.t a trajectory o can be skipped.

Lemma 4.3
If $d(o.l, \tilde{s}.l) \leq \delta$ then $o.l = \arg\min_{r(o)} f_k(r(o), \tilde{s}.l)$.

Proof. Let $r(o)$ $(r(o) \neq o.l)$ be an adjustment of $o.l$. Given $d(o.l, \tilde{s}.l) \leq \delta$, $\mathcal{TC}_k(o.l) = 0 \leq \mathcal{TC}_k(r(o))$. On the other hand, as $d(r(o), o.l) > d(o.l, o.l) = 0$, the snapshot cost $\mathcal{SC}_k(o.l) = 0 < \mathcal{SC}_k(r(o))$. Thus, $r_{opt}(o) = o.l$ if $d(o.l, \tilde{s}.l) \leq \delta$. □

A previous study [17] smooths the distance between each pair of neighboring nodes no matter their relative distances. In contrast, Lemma 4.3 suggests that if a non-seed point remains close to its previous seed point at the current time step, smoothing can be ignored. This avoids over-smoothing close trajectories. Following Example 3.2, $o_2.l = \arg\min_{r(o_2)} f_2(r(o_2), o_1.l)$.

Definition 4.1
A **circle** is given by $\mathcal{Q}(e, x)$, where e is the center and x is the radius.

Definition 4.2
A **segment** connecting two locations l and l' is denoted as $se(l, l')$. The **intersection** of a circle $\mathcal{Q}(e, x)$ and a segment $se(l, l')$ is denoted as $se(l, l') \oplus \mathcal{Q}(e, r)$.

Figure C.3 shows a circle $\mathcal{Q}(o_1.l, \delta)$ that contains $o_1.l$, $o_2.l$, and $o_3.l$. Further, $r(o_6) = se(o_6.l, o_4.l) \oplus \mathcal{Q}(o_4.l, \delta)$.

Lemma 4.4
$se(o.l, \tilde{s}.l) \cap \mathcal{Q}(o.\tilde{l}, \mu \cdot (o.t - o.\tilde{t})) \neq \emptyset$.

Proof. In Section 3.2, we constrain $o.t - o.\tilde{t} \leq \mu \cdot \Delta t$ before smoothing, which implies that $o.l \in \mathcal{Q}(o.\tilde{l}, \mu \cdot (o.t - o.\tilde{t}))$. Hence, $se(o.l, \tilde{s}.l) \cap \mathcal{Q}(o.\tilde{l}, \mu \cdot (o.t - o.\tilde{t})) \neq \emptyset$. □

In Figure C.4, given $o_6.t - o_6.\tilde{t} = 3$, $o_6.l \in se(o_6.l, o_4.l) \cap \mathcal{Q}(o_6.\tilde{l}, 3\mu)$.

Omitting the speed constraint We first show that without utilizing the speed constraint, an optimal adjustment $r_{opt'}(o)$ of $o.l$ that minimizes $f(r(o), \tilde{s}.l)$ can be derived in constant time. Based on this, we explain how to compute r_{opt} based on $r_{opt'}(o)$.

Lemma 4.5
$\forall r'(o) \notin se(o.l, \tilde{s}.l)(\exists r(o) \in se(o.l, \tilde{s}.l)(f_k(r(o), \tilde{s}.l) \leq f_k(r'(o), \tilde{s}.l)))$.

Proof. Let $d(o.l, \tilde{s}.l) = y$. First, we prove that $\forall r'(o) \notin \mathcal{Q}(\tilde{s}.l, y)(\exists r(o) \in se(o.l, \tilde{s}.l)(f_k(r'(o), \tilde{s}.l) \geq f_k(r(o), \tilde{s}.l)))$. Two cases are considered, i.e., (i) $d(r'(o), o.l) \leq y$ and (ii) $d(r'(o), o.l) > y$. For the first case, we can always find an adjustment $r(o) \in se(o.l, \tilde{s}.l)$, such that $d(r'(o), o.l) = d(r(o), o.l)$. Hence, $\mathcal{SC}_k(r'(o)) = \mathcal{SC}_k(r(o))$. However, we have $\mathcal{TC}_k(r'(o)) \geq \mathcal{TC}_k(r(o))$ due to $d(r'(o), \tilde{s}.l) > d(r(o), \tilde{s}.l)$. Thus, $f_k(r'(o), \tilde{s}.l) \geq f_k(r(o), \tilde{s}.l)$. For the second case, it is clear that $\forall r(o) \in se(o.l, \tilde{s}.l)(\mathcal{SC}_k(r'(o)) > \mathcal{SC}_k(r(o)) \wedge \mathcal{TC}_k(r'(o)) \geq \mathcal{TC}_k(r(o)))$. Thus, $f_k(r'(o), \tilde{s}.l) > f_k(r(o), \tilde{s}.l)$.

Second, we prove that $\forall r'(o) \in \mathcal{Q}(\tilde{s}.l, y) \backslash se(o.l, \tilde{s}.l)(\exists r(o) \in se(o.l, \tilde{s}.l)(f_k(r'(o), \tilde{s}.l) \geq f_k(r(o), \tilde{s}.l)))$. We can always find $r(o) \in se(o.l, \tilde{s}.l)$, such that $d(r'(o), \tilde{s}.l) = d(r(o), \tilde{s}.l)$. Hence, $\mathcal{TC}_k(r''(o)) = \mathcal{TC}_k(r(o))$. However, in this case $\mathcal{SC}_k(r'(o)) > \mathcal{SC}_k(r(o))$ due to $r(o) \in se(o.l, \tilde{s}.l) \wedge r'(o) \notin se(o.l, \tilde{s}.l)$. Thus, we have $f_k(r'(o), \tilde{s}.l) > f_k(r(o), \tilde{s}.l)$. □

In Figure C.3, $f(r(o_6), o_4.l) \leq f(r''(o_6), o_4.l)$ and $f(r'(o_6), o_4.l) \leq f(r''(o_6), o_4.l)$ due to $r''(o_6) \notin se(o_4.l, o_6.l)$. Lemma 4.5 indicates that if we ignore the speed constraint in Formula C.8, we can search $r_{opt'}(o)$ just on $se(o.l, \tilde{s}.l)$ without missing any result.

Lemma 4.6
Let $d(r_{opt'}(o), \tilde{s}.l) = b_{opt'} \cdot \delta$. If $d(o.l, \tilde{s}.l) > \delta$ then $b_{opt'} \in \{\mathbf{N}^* \cap [\lambda_1, \lambda_2]\} \cup \lambda_2$, where $\lambda_1 = \frac{d(\tilde{s}.l, o.\tilde{l}) - \mu \cdot (o.t - o.\tilde{t})}{\delta}$, $\lambda_2 = \frac{d(o.l, \tilde{s}.l)}{\delta}$ and $\mathbf{N}^*$ is the natural numbers.

Proof. We start by proving $\max\{\lambda_1, 1\} \leq b_{opt'} \leq \lambda_2$. First, we have $r_{opt'}(o) \in se(o.l, \tilde{s}.l)$ according to Lemma 4.5. Thus, $b_{opt'} \cdot \delta \leq d(o.l, \tilde{s}.l)$, i.e., $b_{opt'} \leq \lambda_2$.

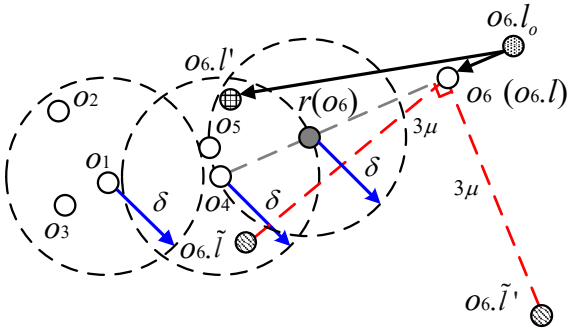


Figure C.4: An example of speed-based pre-processing, i.e., $o_6.l_o \rightarrow o_6$, in Figure C.3 ($\rho = 3$)

Further, $\forall r(o) \in se(o.l, \tilde{s}.l)(d(r(o), \tilde{s}.l) + d(r(o), o.\tilde{l}) \geq d(\tilde{s}.l, o.\tilde{l}))$ due to the triangle inequality. Thus, $b_{opt'} \cdot \delta + \mu \cdot (o.t - o.\tilde{t}) \geq d(\tilde{s}.l, o.\tilde{l})$, i.e., $b_{opt'} \geq \lambda_1$. Moreover, $\forall r'(o) \in se(o.l, \tilde{s}.l)((0 < d(r'(o), \tilde{s}.l) < \delta \wedge d(r(o), \tilde{s}.l) = \delta) \Rightarrow (\mathcal{TC}(r'(o)) = \mathcal{TC}(r(o)) \wedge \mathcal{SC}(r'(o)) > \mathcal{SC}(r(o))))$. Thus, we get $b_{opt'} \geq 1$.

Next, we prove $\forall r'(o), r(o) \in se(o.l, \tilde{s}.l)((b-1) \cdot \delta < d(r'(o), \tilde{s}.l) < b \cdot \delta \wedge d(r(o), \tilde{s}.l) = b \cdot \delta \wedge 1 \leq b \leq \lfloor \lambda_2 \rfloor \wedge b \in \mathbf{N}^*) \Rightarrow (f(r(o), \tilde{s}.l) < f(r'(o), \tilde{s}.l)))$. According to Formula C.5, $\mathcal{TC}(r(o)) = \mathcal{TC}(r'(o))$. Further, $d(o.l, r'(o)) = d(o.l, \tilde{s}.l) - d(r'(o), \tilde{s}.l)$ and $d(o.l, r(o)) = d(o.l, \tilde{s}.l) - b \cdot \delta$. As $d(r'(o), \tilde{s}.l) < b \cdot \delta$ we have $\mathcal{SC}(r(o)) < \mathcal{SC}(r'(o))$. Thus, $b_{opt'} \notin [1, \lfloor \lambda_2 \rfloor] \backslash \mathbf{N}^*$.

Finally, we prove $\forall r'(o) \in se(o.l, \tilde{s}.l)(b \cdot \delta \leq d(r'(o), \tilde{s}.l) \wedge \lfloor \lambda_2 \rfloor < b \leq \lambda_2) \Rightarrow (f(o.l, \tilde{s}.l) < f(r'(o), \tilde{s}.l)))$. Similar to the above proof, in this case $\mathcal{TC}(r'(o)) = \mathcal{TC}(o.l) \wedge \mathcal{SC}(r'(o)) > \mathcal{SC}(o.l)$. Thus, $b_{opt'} \notin (\lfloor \lambda_2 \rfloor, \lambda_2] \backslash \{\lambda_2\}$. □

In Figure C.3, we have $r_{opt'}(o_6) \in \{o_6.l, r(o_6), r'(o_6)\}$. Based on Lemmas 4.3 to 4.6, we let $d(r(o), \tilde{s}.l) = b \cdot \delta$ and simplify Formula C.8 to the following function:

$$f_k(b, \tilde{s}.l) = (d(o.l, \tilde{s}.l) - b \cdot \delta)^2 + \alpha \cdot (\delta \cdot (b-1))^2 \quad s.t.\ b \in \{\mathbf{N}^* \cap [\lambda_1, \lambda_2]\} \cup \lambda_2, \tag{C.9}$$

where $\lambda_1 = \frac{d(\tilde{s}.l, o.\tilde{l}) - \mu \cdot (o.t - o.\tilde{t})}{\delta}$ and $\lambda_2 = \frac{d(o.l, \tilde{s}.l)}{\delta}$. The snapshot cost $(d(o.l, \tilde{s}.l) - b \cdot \delta)^2$ is derived according to Lemma 4.5, i.e., $o.l, r_{opt'}(o)$ and $\tilde{s}.l$ are on the same line segment; while the historical cost $(\delta \cdot (b-1))^2$ is obtained by simply plugging $d(r(o), \tilde{s}.l) = b \cdot \delta$ into Formula C.8. The objective function in Formula C.9 is a continuous. Thus, the $b_{opt'}$ ($\in \{\mathbf{N}^* \cap [\lambda_1, \lambda_2]\} \cup \lambda_2$) that minimizes the function can be obtained in constant time without sacrificing accuracy.

Example 4.1
Continuing Example 3.2 and given $d(o_6.l, o_4.l) = 25$, $\alpha = 2.1$ and $\delta = 10$, we get $b_{opt'} = 1$ and $r_{opt'}(o_6) = r(o_6)$.

Introducing the speed constraint Recall that $r_{opt'}(o)$ is the optimal adjustment of $o.l$ without taking the speed constraint in Formula C.8 into account, while $r_{opt}(o)$ takes the constraint into account. We have narrowed the range of $r_{opt'}$ to a set of discrete locations on $se(o.l, \tilde{s}.l)$ without sacrificing any accuracy. Further, if $r_{opt'} \in \mathcal{Q}(o.\tilde{l}, \mu \cdot \Delta t)$ then $r_{opt}(o) = r_{opt'}(o)$. However, if $r_{opt'} \notin \mathcal{Q}(o.\tilde{l}, \mu \cdot \Delta t)$, $r_{opt'}(o)$ is an invalid adjustment. In this case, letting $d(r_{opt}(o), \tilde{s}.l) = b_{opt} \cdot \delta$, we propose to approximate r_{opt} by searching only in the narrowed range of $r_{opt'}$, i.e., we propose to compute b_{opt} approximately as follows.

$$\begin{aligned} & b_{opt} = \arg \min_{b \in \{\mathbb{N}^* \cap [\lambda_1, \lambda_2]\} \cup \lambda_2} |b - b_{opt'}| \\ & s.t.\ \mathcal{Q}(o.\tilde{l}, \mu \cdot (o.t - o.\tilde{t})) \cap \mathcal{Q}(\tilde{s}.l, b \cdot \delta) \neq \emptyset, \end{aligned} \tag{C.10}$$

where $\mathcal{Q}(o.\tilde{l}, \mu \cdot (o.t - o.\tilde{t})) \cap \mathcal{Q}(\tilde{s}.l, b \cdot \delta) \neq \emptyset$ indicates that $r_{opt}(o) \in \mathcal{Q}(o.\tilde{l}, \mu \cdot (o.t - o.\tilde{t}))$ must hold due to $d(r_{opt}(o), \tilde{s}.l) = b_{opt} \cdot \delta$. After getting $b_{opt'}$, b_{opt} can be located according to $se(o.l, \tilde{s}.l) \oplus \mathcal{Q}(o.\tilde{l}, \mu \cdot (o.t - o.\tilde{t}))$ in constant time. Following Example 4.1 and given $o_6.t - o_6.\tilde{t} = 3$ and $\mu = 9$, $r_{opt}(o_6) = r(o_6)$ if $o_6.l_1 = o_6.\tilde{l}$, while $r_{opt}(o_6) = o_6.l$ if $o_6.l_1 = o_6.\tilde{l}'$ (shown in Figure C.4). Specifically, in the latter case, $o_6.l$ is the only feasible solution of $r_{opt}(o_6)$ according to Formula C.10, as $se(o_6.l, o_4.l) \oplus \mathcal{Q}(o_6.\tilde{l}', 3\mu) = o_6.l$. Note that computing r_{opt} using Formula C.10 may not yield an optimal value that minimizes $f_k(r(o), \tilde{s}.l)$. This is because we approximate the feasible region of r_{opt} by the narrowed range of $r_{opt'}$ and may miss an $r(o)$ ($r(o) \in \mathcal{Q}(o.\tilde{l}, \mu \cdot (o.t - o.\tilde{t})) \backslash se(o.l, \tilde{s}.l)$) that minimizes Formula C.8. However, experiments show that $r_{opt'} = r_{opt}$ in most case. The underlying reasons are that the maximum distance a trajectory can move under the speed limitation is generally far larger than the distance a trajectory actually moves between any two time steps and that we constrain $d(o.l, o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t})$ before smoothing, which "repairs" large noise to some extent. So far, the efficiency of computing $r_{opt}(o)$ using Formula C.8 has been improved to $O(1)$ time complexity.

4.2 Shifting of Seed Points

Section 4.1 assumes that the previous seed point $\tilde{s}.l$ evolves gradually when smoothing $o \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k \backslash \{\tilde{s}\}$ at dt_k, which is not always true. Thus, $\tilde{s}.l$ may also need to be smoothed. We first select a "pivot" for smoothing $\tilde{s}.l$. An existing method [30] maps the noise point to the accurate point that is closest to it in a batch mode. Inspired by this, we smooth $\tilde{s}.l$ using $o.l$ ($o \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k$), which is a set of discrete locations. This is based on the observation that the travel companions of each trajectory evolves gradually due to the smooth movement of trajectories. Next, we determine which trajectory o should be selected as a "pivot" to smooth $\tilde{s}.l$.

Evolutionary clustering assigns a low cost (cf. Formula C.1) if the clus-

terings change smoothly during a short time period. Since we use cost embedding, we consider the location of a trajectory o as evolving smoothly if the distance between o and o' ($o' \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k \backslash \{o\}$) varies only little between two adjacent time steps. This is essentially evaluated by $f_k(o)$ (cf. Formula C.7), which measures the cost of smoothing o' ($o' \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k \backslash \{o\}$) according to o. Hence, we select the "pivot" for smoothing $\tilde{s}.l$ using the following formula:

$$\tilde{s}_{new} = \arg \min_{o \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k} f_k(o) \tag{C.11}$$

After obtaining a "pivot" $\tilde{s}_{new}$, instead of first smoothing $\tilde{s}.l$ according to $\tilde{s}_{new}.l$ and then smoothing $o.l (o \in \mathcal{M}_{k-1}(\tilde{s}) \backslash \{\tilde{s}\})$ by $\tilde{s}.l$, we shift the seed point of $o \in \mathcal{M}_{k-1}(\tilde{s}) \backslash \{\tilde{s}\}$ from $\tilde{s}$ to $\tilde{s}_{new}$ and use $\tilde{s}_{new}$ to smooth other trajectories o ($o \neq \tilde{s}_{new}$). The reasons are: (i) by Formula C.11, $\tilde{s}_{new}$ is the trajectory with the smoothest movement from dt_{k-1} to dt_k among trajectories in $\mathcal{M}_{k-1}(\tilde{s})$, and thus it is less important to smooth it; (ii) we can save $|\mathcal{M}_{k-1}(\tilde{s})| - 1$ computations in Formula C.7. Formula C.11 suggests that the seed point may not be shifted, i.e., $\tilde{s}_{new}$ may be $\tilde{s}$. Intuitively, when computing $\tilde{s}_{new}$, the locations of all trajectories in their corresponding minimal group are smoothed; and with the seed point shifting strategy, smoothing does not require that the previous seed point is active at the current time step.

Example 4.2
Continuing Example 4.1, given $f_2(o_6.l) = \min\{f_2(o_4.l), f_2(o_5.l), f_2(o_6.l)\}$, $\tilde{s}_{new} = \tilde{s} = o_6$. When calculating $\tilde{s}_{new}$, we get $r_{opt}(o_4) = o_4.l$, $r_{opt}(o_5) = o_5.l$, and $r_{opt}(o_6) = r(o_6)$.

The time complexity of smoothing a minimal group $\mathcal{M}_{k-1}(\tilde{s})$ is $O(|\mathcal{M}_{k-1}(\tilde{s})|^2)$.

4.3 Speed-based Pre-processing

We present the pre-processing that forces each to-be-smoothed trajectory $o \in \mathcal{O}_k \cap (\bigcup_{\tilde{s} \in \mathcal{S}_{k-1}} \mathcal{M}_{k-1}(\tilde{s}))$ to observe the speed constraint. The pre-processing guarantees the correctness of the normalization of the snapshot and historical costs and can repairs large noise to some extent. We denote the location of o before pre-processing as $o.l_o$ and the possible location after as $o.l_p$.

A naive pre-processing strategy is to map $o.l_o$ to a random location on or inside $\mathcal{Q}(o.\tilde{l}, \mu \cdot \Delta t)$. However, this random strategy may make the smoothing less reasonable.

Example 4.3
Continuing Example 4.1, Figure C.4 shows two possible locations $o_6.l$ and $o_6.l'$ of $o_6.l_o$, both of which are chosen at random while observing the speed constraint, i.e., they are located on and inside $\mathcal{Q}(o_6.\tilde{l}, 3\mu)$, respectively. Since $d(o_6.l', o_4.l) < \delta < d(o_6.l, o_4.l)$, the adjustment of $o_6.l'$ is $o_6.l'$ itself while that of $o_6.l$ is $r(o_6)$.

In this example, $o_6.l'$ is less reasonable than $r(o_6)$. Specifically, according to the minimum change principle [30], the changes to the data distribution made by the speed-based pre-processing and the neighbor-based smoothing should be as small as possible. However, considering $d(o_6.l_o, o_4.l)$, $o_6.l'$ is too close to $o_4.l$ compared with $o_6.l$ and $r(o_6)$. Given a pre-processed location $o.l$, its change due to smoothing has already been minimized through Formula C.8. Thus, to satisfy the minimum change principle, we just need to make the impact of speed-based pre-processing on neighbor-based smoothing as small as possible. Hence, we find the pre-processed $o.l$ via the speed constraint as follows.

$$\begin{aligned} o.l &= \arg\min_{o.l_p} \left| d(o.l_p, \tilde{s}.l) - d(o.l_o, \tilde{s}.l) \right| \\ s.t.\, & d(o.l_p - o.\tilde{l}) \leq \mu \cdot (o.t - o.\tilde{t}) \end{aligned} \tag{C.12}$$

This suggests that the difference between $d(o.l, \tilde{s}.l)$ and $d(o.l_o, \tilde{s}.l))$ is expected to be as small as possible, in order to mitigate the effect of speed-based pre-processing on computing historical cost. Before applying Formula C.12, we pre-process $\tilde{s}.l$ so that it also follows the speed constraint:

$$\begin{aligned} \tilde{s}.l &= \arg\min_{\tilde{s}.l_p} d(\tilde{s}.l_o, \tilde{s}.l_p) \\ s.t.\, & d(\tilde{s}.l_p - \tilde{s}.\tilde{l}) \leq \mu \cdot (\tilde{s}.t - \tilde{s}.\tilde{t}) \end{aligned} \tag{C.13}$$

As $\tilde{s}$ is not smoothed by any trajectories in $\mathcal{M}_{k-1}(\tilde{s})$ (cf. Section 4.1), Formula C.13 lets the closest location to $\tilde{s}.l_p$ satisfying the speed constraint be $\tilde{s}.l$. This is also in accordance with the minimum change principle [30]. According to the seed point shifting strategy, we examine each $o \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k$ to identify the most smoothly moving trajectory as $\tilde{s}_{new}$. Thus, before this process, we have to force each $o \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k$ to follow the speed constraint w.r.t. the current to-be-examined seed point $\tilde{s}$, i.e., computing $o.l$ w.r.t. $\tilde{s}.l$ according to Formula C.12. Obviously, a speed-based pre-processing is only needed when $d(o.l_o - o.\tilde{l}) > \mu \cdot (o.t - o.\tilde{t})$; otherwise, $o.l = o.l_o$. Formulas C.12 and C.13 can be computed in constant time.

5 Algorithms

We first introduce a grid index and then present the algorithms for generating minimal groups, smoothing locations, and performing the clustering, together with a set of optimization techniques.

5.1 Grid Index

We use a grid index [13] to accelerate our algorithms. Figure C.3 shows an example index. Specifically, the diagonal of each grid cell (denoted as g) has

Algorithm 4: Generating minimal groups

Input: a set of trajectories $\mathcal{O}_k$, a threshold δ
Output: a set of seed point $\mathcal{S}_k$ and minimal groups $\bigcup_{s\in\mathcal{S}_k}\mathcal{M}_k(s)$

```
for each o ∈ O_k do
    S_k ← S_k ∪ o if ∀s ∈ S_k(d(o.l, s.l) > δ)
for each o ∈ O_k \ S_k do
    M_k(s) ← M_k(s) ∪ o
    s ← arg min_{s∈I_δ(g(o))∧d(o.l,s.l)≤δ} d(o.l, s.l)
return S_k and ⋃_{s∈S_k} M_k(s)
```

length ε, which is the parameter used in DBSCAN [13]. This accelerates the process of finding core points. The number of trajectories that fall into g is denoted as $|g|$. Given $o \in g$ at dt_k and $o \in \mathcal{M}_k(s)$, $\mathcal{G}(g)$ is the collection of grid cells g', such that $o' \in \mathcal{M}_k(s) \wedge o' \in g'$. Following Example 3.1, $\mathcal{G}(g_{34}) = \{g_{24}, g_{34}\}$, as shown in Figure C.3. The smallest distance between the boundaries of two grid cells, g and g', is denoted as $\min(g, g')$. Clearly, $\min(g, g) = 0$. For example in Figure C.3, $\min(g_{24}, g_{44}) = \varepsilon$. Next, we introduce the concept of h-closeness [13].

Definition 5.1
Two grid cells g and g' are ***h-close***, if $\min(g, g') \leq h$. The set of the ***h-close*** grid cells of g is denoted as $\mathcal{I}_h(g)$.

Lemma 5.1
For $o \in g$, we have $d(o.l, o'.l) > h$ if $o' \in g' \wedge g' \notin \mathcal{I}_h(g)$.

The proof is straightforward. We utilize two distance parameters, i.e., ε for clustering (cf. Definition 2.6) and δ for finding minimal groups (cf. Definition 3.2). Thus, we only need to consider $\mathcal{I}_h(g)$, where $h = \varepsilon, \delta$. Following again existing work [13], we define $\mathcal{I}_\varepsilon(g_{ij}) = \Omega \backslash (g_{i_1 j_1} \cup g_{i_1 j_2} \cup g_{i_2 j_1} \cup g_{i_2 j_2})$, where $\Omega = \{g_{i'j'} | i_1 \leq i' \leq i_2 \wedge j_1 \leq j' \leq j_2\}$ and $i_1 = i - 2$, $i_2 = i + 2$, $j_1 = j - 2$ and $j_2 = j + 2$. For example in Figure C.3, $\mathcal{I}_\varepsilon(g_{44}) = \Omega \backslash (g_{22} \cup g_{62} \cup g_{26} \cup g_{66})$, where $\Omega = \{g_{ij} | 2 \leq i, j \leq 6\}$. Since we set $\delta < \varepsilon$, we only need to compute $\mathcal{I}_h(g)$, such that $h < \varepsilon$.

Lemma 5.2
$\mathcal{I}_h(g_{ij}) = \mathcal{I}_\varepsilon(g_{ij})$ if $\frac{\varepsilon}{\sqrt{2}} \leq h < \varepsilon$; otherwise $\mathcal{I}_h(g_{ij}) = \{g_{i'j'} | i - 1 \leq i' \leq i + 1 \wedge j - 1 \leq j' \leq j + 1\}$.

The proof of Lemma 5.2 follows from the grid cell width being $\frac{\varepsilon}{\sqrt{2}}$. In Figure C.3, given $\frac{\varepsilon}{\sqrt{2}} \leq h < \varepsilon$, $\mathcal{I}_\delta(g_{44}) = \mathcal{I}_\varepsilon(g_{44})$.

5.2 Generating Minimal Groups

Sections 3 and 4 indicate that o is smoothed at dt_k if $\exists \mathcal{M}_{k-1}(\tilde{s})(o \in \mathcal{M}_{k-1}(\tilde{s}))$; otherwise, o is considered as an "outlier," to which neighbor-based smoothing cannot be applied. Thus, we aim to include as many trajectories as possible in the minimal groups, in order to smooth as many trajectories as possible.

According to the above analysis, an optimal set of minimal groups should satisfy $\forall o \in \mathcal{O}_k \backslash (\bigcup_{s \in \mathcal{S}_k} \mathcal{M}_k(s)), \forall s \in \mathcal{S}_k \ (d(o,s) > \delta)$. Clearly, $\bigcup_{s \in \mathcal{S}_k} \mathcal{M}_k(s)$, the set of trajectories in the minimal groups, is determined given $\mathcal{S}_k$. Definition 3.2 implies that the local density of a seed point s should be not small, i.e., $|\mathcal{N}_\delta(s)| \geq \rho$. It guarantees that there is at least one trajectory $o\,(=s)$ in a minimal group that is not located at the border of a cluster. Considering the above requirement and constraint, we have to enumerate all the possible combinations to get the optimal set of $\mathcal{S}_k$, which is infeasible.

Therefore, we propose a greedy algorithm, shown in Algorithm 4, for computing a set of minimal groups at dt_k. Each trajectory o is mapped to a grid cell g before generating minimal groups. We first greedily determine $\mathcal{S}_k$ and then generate minimal groups according to $\mathcal{S}_k$. This is because a non-seed point o attached to a minimal group $\mathcal{M}_k(s)$ at the very beginning may turn out to be closer to another newly obtained seed point s'. This incurs repeated processes for finding a seed point for o. Instead, we compute the seed point s for each $o \in \mathcal{O}_k$ exactly once. According to Lemma 5.1, we will not miss any possible seed point for o by searching $\mathcal{I}_\delta(o)$ rather than $\mathcal{S}_k$ (Line 5). Note that Algorithm 4 generates minimal groups $\mathcal{M}_k(s)$ such that $|\mathcal{M}_k(s)| < \rho$. We simply ignore these during smoothing.

5.3 Evolutionary Clustering

Smoothing Algorithm 5 gives the pseudo-code of the smoothing algorithm. We maintain *glp* to record the current minimal $f_k(o'.l) = \sum_{o \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k \backslash \{o'\}} f_k(r(o), o'.l)$ $(o' \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k)$ and maintain $\mathcal{A}$ to record adjustments w.r.t. $o'.l$ (Line 1). The computation of $f_k(o'.l)$ is terminated early if its current value exceeds *glp* (Lines 8–9). As can be seen, if o' $(o' \in \mathcal{M}_{k-1}(\tilde{s})) \cap \mathcal{O}_k$ is identified as the trajectory with the smoothest movement in its minimal group, the set of adjustments $r(o)$ $(o \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k)$ w.r.t $o'.l$ is returned, i.e., $\mathcal{R}_k(\mathcal{M}_{k-1}(\tilde{s}))$.

Optimizing modularity Modularity is a well-known quality measure for clustering [17, 37], which is computed as follows.

$$QS = \sum_{c \in \mathcal{C}_k} \left(\frac{IS(c)}{TS} - \left(\frac{DS(c)}{TS} \right)^2 \right) \tag{C.14}$$

Here, TS is the sum of similarities of all pairs of trajectories, $IS(c)$ is the sum of similarities of all pairs of trajectories in cluster c, $DS(c)$ is the sum

Algorithm 5: Smoothing

Input: a minimal group $\mathcal{M}_{k-1}(\tilde{s})$
Output: a set of adjustments $\mathcal{R}_k(\mathcal{M}_{k-1}(\tilde{s}))$

$sum \leftarrow 0$, $\mathcal{R}_k(\mathcal{M}_{k-1}(\tilde{s})) \leftarrow \emptyset$, $glp \leftarrow \infty$, $\tilde{s}_{new} \leftarrow null$, $\mathcal{A} \leftarrow \emptyset$
for *each* $o' \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k$ **do**
  compute $o'.l$ according to Formula C.13, $\mathcal{A} \leftarrow o'.l$, $sum \leftarrow 0$
  for *each* $o \in \mathcal{M}_{k-1}(\tilde{s}) \cap \mathcal{O}_k \backslash \{o'\}$ **do**
    compute $o.l$ according to Formula C.12
    compute $r_{opt}(o)$ according to Formulas C.9 and C.10
    $sum \leftarrow sum + f_k(r_{opt}(o), o'.l)$
    if $sum \geq glp$ **then**
      break /* o' must not be $\tilde{s}_{new}$ */
    $\mathcal{A} \leftarrow \mathcal{A} \cup r_{opt}(o)$
  if $|\mathcal{A}| = |\mathcal{M}_{k-1}(\tilde{s})|$ **then**
    $glp \leftarrow sum$, $\tilde{s}_{new} \leftarrow o'$, $\mathcal{R}_k(\mathcal{M}_{k-1}(\tilde{s})) \leftarrow \mathcal{A}$
return $\mathcal{R}_k(\mathcal{M}_{k-1}(\tilde{s}))$

of similarities between a trajectory in cluster c and any trajectory in cluster c' ($c' \in \mathcal{C}_k \backslash \{c\}$). A high QS indicates a good clustering result. The similarity between any two trajectories o and o' is defined as $\frac{1}{d(o.l, o'.l)}$.

A previous study [17] iteratively adjusts ε to find the (local) optimal QS as well as the clustering result at each time step. Specifically, given an ε, a constant $\Delta\varepsilon$, and the current clustering result $\mathcal{C}_k$, it calculates three modularity during each iteration: QS_h, QS_l, and QS_m. $QS_m = QS$ is the modularity of $\mathcal{C}_k$. QS_h is calculated from pairs in $\mathcal{C}_k$ with a similarity in the range $[\varepsilon, \varepsilon + \Delta\varepsilon]$, and QS_l is calculated from pairs in $\mathcal{C}_k$ with a similarity in the range $[0, \varepsilon - \Delta\varepsilon)$. Then ε is adjusted as follows.

- If $QS_h = \max\{QS_h, QS_l, QS_m\}$, ε increases by $\Delta\varepsilon$;
- If $QS_l = \max\{QS_h, QS_l, QS_m\}$, ε decreases by $\Delta\varepsilon$;
- If $QS_m = \max\{QS_h, QS_l, QS_m\}$, ε is unchanged.

The first two cases leads to another iteration of calculating QS_h, QS_l, and QS_m using the newly updated ε, while the last case terminates the processing. This iterative optimization of modularity [17] has a relatively high time cost. We improve the cost by only updating ε at dt_k (denoted as ε_k) once to "approach" the (local) optimal modularity of $\mathcal{C}_k$. Although ε_k is then used for clustering at dt_{k+1} instead of at dt_k, the quality of the clustering is generally still improved, as the clustering result evolves gradually. Specifically, ε is still obtained by the iterative optimization at the first time step.

Algorithm 6: Evolutionary clustering (ECO)

Input: a snapshot $\mathcal{O}_k$, a clustering result $\mathcal{C}_{k-1}$, a set of minimal groups $\bigcup_{s \in \mathcal{S}_{k-1}} \mathcal{M}_{k-1}(\tilde{s})$, thresholds δ, ρ, α, ε_{k-1}, $\Delta\varepsilon$, *minPts*
Output: a clustering result C_k

1 smooth $\mathcal{O}_k$ and get the set of adjustments $\mathcal{R}_k$ /* Algorithm 5 */
2 build a grid index according to ε_{k-1}
3 generate minimal groups based on $\mathcal{R}_k$ /* Algorithm 4 */
4 cluster $\mathcal{R}_k$ to get $\mathcal{C}_k$ /* DBSCAN */
5 update ε_{k-1} to ε_k according to $\mathcal{C}_k$
6 map $c \in \mathcal{C}_k$ to $c' \in \mathcal{C}_{k-1}$ /* literature [17]*/
7 **return** $\mathcal{C}_k$

Grid index and minimal group based accelerations As we set the grid cell width to $\frac{\varepsilon}{\sqrt{2}}$, each $o \in g$ is a core point if $|g| \geq \mathit{minPts}$ [13]. Similarly, if $|\mathcal{M}_k(s)| \geq \rho$, s is a core point. These efficient checks accelerate the search for core points as well as DBSCAN. In Example 4.1 and given *minPts*=3, o_4, o_5 and o_6 are core points due to $|g_{44}| = 3$ and o_1 is core point due to $|\mathcal{M}_2(o_1)| = 3$.

Evolutionary clustering of streaming trajectories All pieces are now in place to present the algorithm for evolutionary clustering of streaming trajectories (ECO), shown in Algorithm 6. The sub-procedures in lines 1–5 are detailed in the previous sections. Note that, as we perform evolutionary clustering at each time step with an updated ε, a grid index is built at each time step once locations arrive. The time cost of this is neglible [19]. Also note that, the grid index should be built after smoothing, as the locations of trajectories are changed. Finally, we connect clusters in adjacent time steps with each other (Line 6) as proposed in the literature [17]. This mapping aims to find the evolving, forming, and dissolving relationships between $c'_k \in \mathcal{C}_{k-1}$ and $c'_k \in \mathcal{C}_k$. Building on Examples 2.1 and 3.2, $c_1 \in \mathcal{C}_1$ evolves to $c_1 \in \mathcal{C}_2$ while $c_2 \in \mathcal{C}_1$ evolves to $c_2 \in \mathcal{C}_2$, and no clusters form or dissolve. The details of the mapping are available elsewhere [17]. The time complexity of ECO at time step dt_k is $O(|\mathcal{O}_k|^2)$.

6 Experiments

We report on extensive experiments aimed at achieving insight into the performance of ECO.

Table C.2: Parameter ranges and default values

Parameter	Range
minPts	2, 4, 5, 6, 7, **8**, 10
δ	200, 300, **400**, 500, **600**, 700, 800
α	0.1, 0.3, 0.5, 0.7, **0.9**
ρ	4, 5, **6**, 7, 8

6.1 Experimental Design

Datasets. Two real-life datasets, Chengdu (CD) and Hangzhou (HZ), are used. The CD dataset is collected from 13,431 taxis over one day (Aug. 30, 2014) in Chengdu, China. It contains 30 million GPS records. The HZ dataset is collected from 24,515 taxis over one month (Nov. 2011) in Hangzhou, China. It contains 107 million GPS records. The sample intervals of CD and HZ are 10s and 60s.

Comparison algorithms and experimental settings. We compare with three methods:

- Kim-Han [17] is a representative density-based evolutionary clustering method. It evaluates costs at the individual distance level to improve efficiency.
- DYN [37] is the state-of-the-art evolutionary clustering. It adapts a particle swarm algorithm and random walks to improve result quality.
- OCluST [27] is the state-of-the-art for traditional clustering of streaming trajectories that disregards the temporal smoothness. It continuously absorbs newly arriving locations and updates representative trajectories maintained in a novel structure for density-based clustering.

In the experiments, we study the effect on performance of the parameters summarized in Table C.2. $\Delta\varepsilon$ is set to 50 on both datasets. The number of generations and the population size of DYN [37] are both set to 20, in order to be able to process large-scale streaming data. Other parameters are set to their recommended values [17, 27, 37]. We compare with Kim-Han [17] and DYN [37] because, to the best of our knowledge, no other evolutionary clustering methods exist for trajectories. To adapt these two to work on clustering trajectories, we construct a graph on top of the GPS data by adding an edge between two locations (nodes) $o.l$ and $o'.l$ if $d(o.l, o'.l) \leq \tau$, where $\tau = 3000$ on CD and $\tau = 1000$ on HZ. All algorithms are implemented in C++, and the experiments are run on a computer with an Intel Core i9-9880H CPU (2.30 GHz) and 32 GB memory.

Performance metrics. We adopt *modularity QS* (cf. Formula C.14) to measure the quality of clustering. We report *QS* as average values over all time steps.

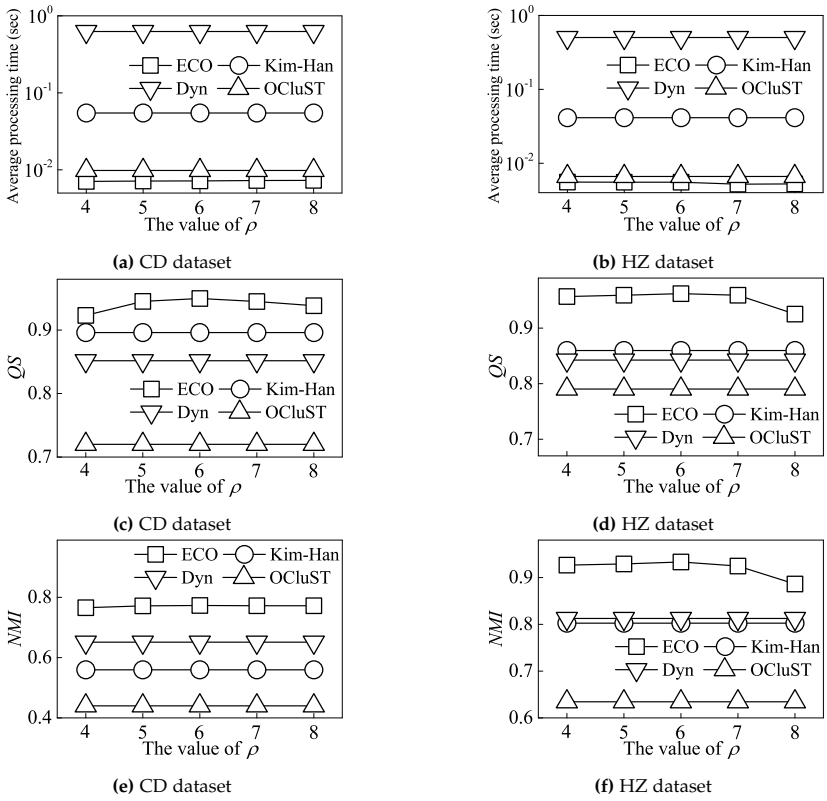


Figure C.5: Effects of varying ρ

The higher the QS, the better the clustering. Moreover, we use *normalized mutual information NMI* [31] to measure the similarity between two clustering results obtained at consecutive time steps.

$$NMI = \frac{I(\mathcal{C}_{k-1}; \mathcal{C}_k)}{\sqrt{H(\mathcal{C}_{k-1}) \cdot H(\mathcal{C}_k)}}, \tag{C.15}$$

where $I(\mathcal{C}_{k-1}; \mathcal{C}_k)$ is the mutual information between clusters $\mathcal{C}_{k-1}$ and $\mathcal{C}_k$, $H(\mathcal{C}_k)$ is the entropy of cluster $\mathcal{C}_k$. Specifically, the reported *NMI* values are averages over all time steps. Clusters evlove more smoothly if *NMI* is higher. Finally, efficiency is measured as the *average processing time* per record at each time step.

6.2 Comparison and Parameter Study

We study the effect of parameters (summarized in Table C.2) on the performance of the four methods.

Effects of varying ρ Figure C.5 reports on the effect of varying δ. First, ECO generally outperforms the baselines in terms of all performance metrics. In particular, ECO's average processing time is almost one order of magnitude lower than Kim-Han and almost two orders of magnitude better than that of Dyn on both datasets. Moreover, ECO is even slightly more efficient than OCluST, because the latter updates its data structure repeatedly for macro-clustering. The high efficiency and quality of ECO are mainly due to three reasons: (i) Except for the initialization, ECO excludes iterative processes and is accelerated by grid indexing and the proposed optimizing techniques; (ii) ECO takes into account temporal smoothness, which is designed specifically for trajectories (cf. Formula C.6); (iii) Locations with the potential to incur mutation of a clustering are adjusted to be closer to their neighbors that evolve smoothly, generally increasing the intra-density and decreasing the inter-density of clustering.

Second, we consider the effects of varying ρ. Figures C.5a and C.5b show that the average processing time is relatively stable. This is because the most time-consuming process in ECO is the clustering, which depends highly on the volume of data arriving at each time step. All four methods achieve higher efficiency on HZ than CD, due to CD's larger average data size of each time step. Figures C.5c–C.5f show that as ρ grows, *QS* and *NMI* first increase and then drop. On the one hand, trajectories with high local density are generally more stable, i.e., more likely to remain in the same cluster in adjacent time steps. On the other hand, with a too large ρ, few minimal groups are generated, and thus few locations are smoothed. As the baselines do not have parameter ρ, their performance is unaffected.

Figure C.6 reports in the effects of varying δ. Specifically, ECO outperforms the baselines in terms of all metrics, and the processing times of the methods remain stable, as shown in Figures C.6a and C.6b. Figures C.6c–C.6f indicate that as δ increases, both *QS* and *NMI* first increase and then drop. On the one hand, a too small δ leads to a small number of trajectories forming minimal groups and being smoothed; on the other hand, a too large δ also leads to few smoothing operations, as more pairs of trajectories o and o' ($o, o' \in \mathcal{M}_{k-1}(\tilde{s})$) satisfy $d(o.l, o'.l) \leq \delta$ at dt_k. Since the baselines do not utilize parameter δ, they are unaffected by variations in δ.

Effects of varying *minPts* Figure C.7 shows the effect of *minPts* on clustering. When varying *minPts*, the effects are similar to those seen when varying ρ and δ. Figures C.7c and C.7d show that *QS* of ECO, Dyn, and OCluST drop as *minPts* increases. The findings indicate that the average distance between trajectories in different clusters decreases with *minPts*. Assume that o is a core point when *minPts* is small and that o' is a border point that is density reachable from o. As *minPts* increases, o may no longer be a core point. In this

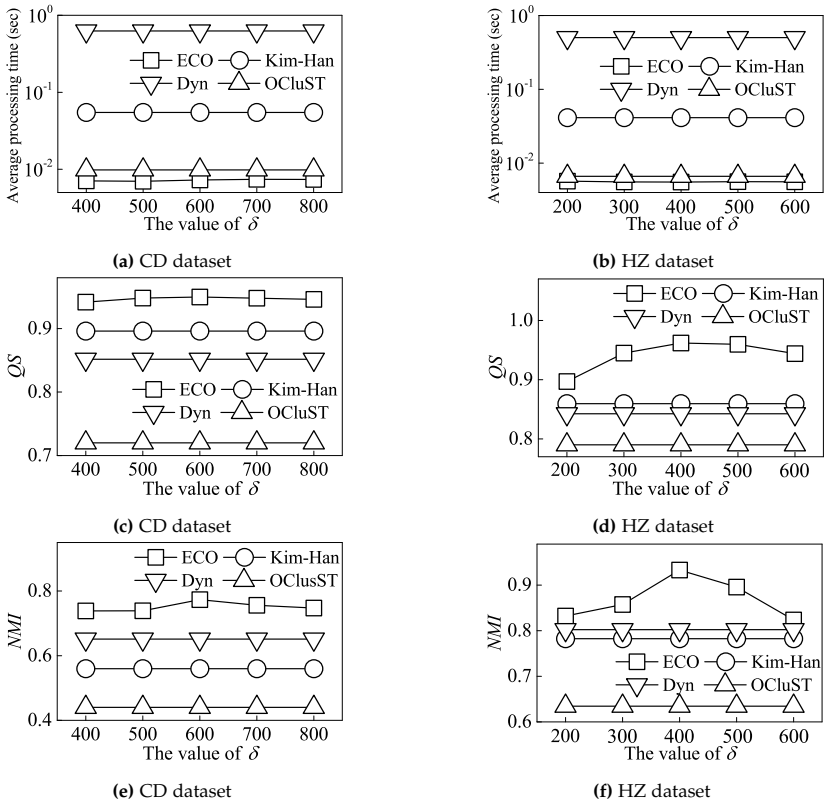


Figure C.6: Effects of varying δ

case, o' and o may be density reachable from different core points and may thus be in different clusters, even if $d(o.l, o'.l) \leq \varepsilon$. As a result, the distances between trajectories in different clusters decrease. Figures C.7e and C.7f show that *NMI* increases with *minPts* for ECO, Dyn, and OCluST. The findings suggest that with a smaller *minPts*, the trajectories at the "border" of a cluster are more likely to shift between being core points and being non-core points over time. In this case, clusters fluctuate more between consecutive time steps for a smaller *minPts*. As Dyn adopts particle swarm clustering instead of density based clustering, Dyn is unaffected by *minPts*.

Effects of varying α Figure C.9 shows the effects of varying α. First, ECO always achieves the best performance among all methods. Second, all methods exhibit stable performance when varying α. Third, both *QS* and *NMI* of ECO increase with α. On the one hand, according to Formula C.8, a larger α generally leads to a smaller distance between two trajectories in the same cluster. On the other hand, Formula C.8 reduces the historical cost as α

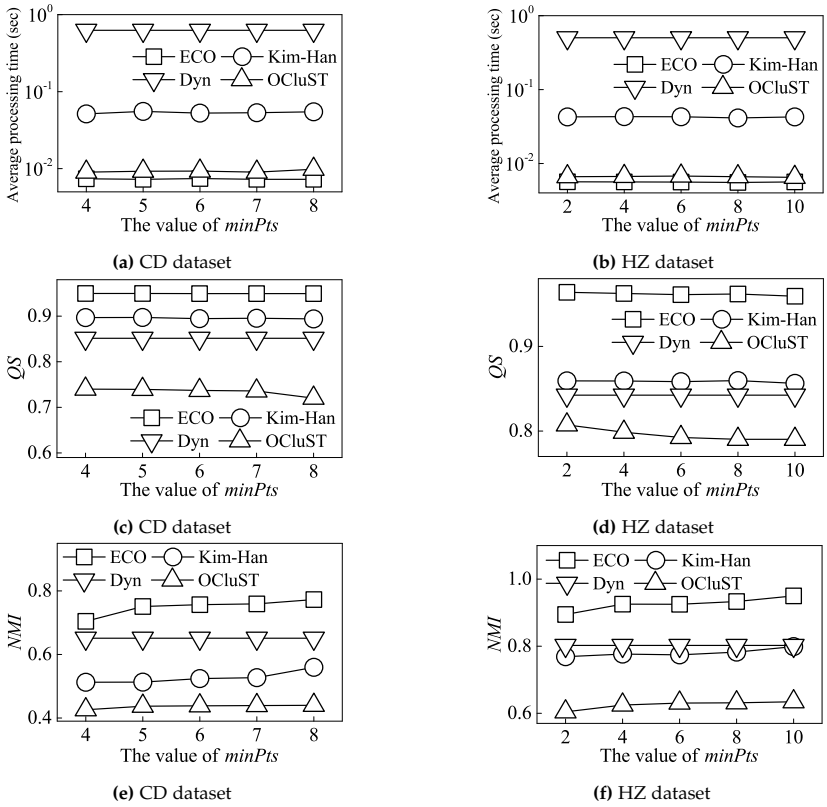


Figure C.7: Effects of varying *minPts*

increases, which renders the evolution of clusters more smooth.

6.3 Scalability

To study the scalability, we vary the data size from 20% to 100%, which is done by randomly sampling moving objects by their IDs. The results are reported in Figure B.6. First, ECO achieves the highest efficiency for large datasets, but is less efficient than OCluST for small datasets. This is because the locations of trajectories are generally distributed uniformly when data size is small. In this case, the grid index becomes less useful. As expected, the processing times of all methods increase with the dataset size.

Second, *QS* improves with the data size for all methods, with ECO always being best. As illustrated above, data distribution is generally non-uniform for a 100% dataset and becomes increasingly uniform as the data size decreases. Thus, the average distances between any pair of trajectories in the same cluster become smaller as data size increases, resulting in a larger *QS*.

Third, ECO achieves the highest *NMI*, which increases with the data size.

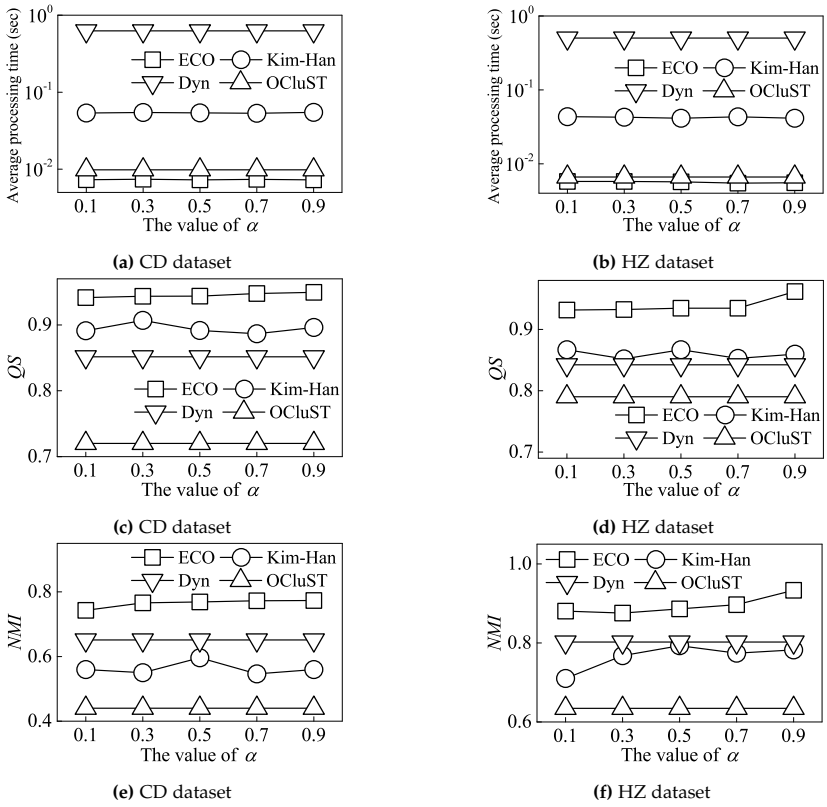


Figure C.8: Effects of varying α

This is mainly because fewer trajectories are able to form minimal groups and be subjected to smoothing when the average distances between any pair of trajectories increases.

7 Related Work

We proceed to review related works on streaming trajectory clustering and evolutionary clustering.

7.1 Streaming Trajectory Clustering

Streaming trajectory clustering finds representative paths or common movement trends among objects in real time. The main target of existing studies of streaming trajectory clustering is to update clusters continuously with high efficiency. Jensen et al. [16] exploit an incrementally maintained clustering feature CF and propose a scheme for continuous clustering of moving

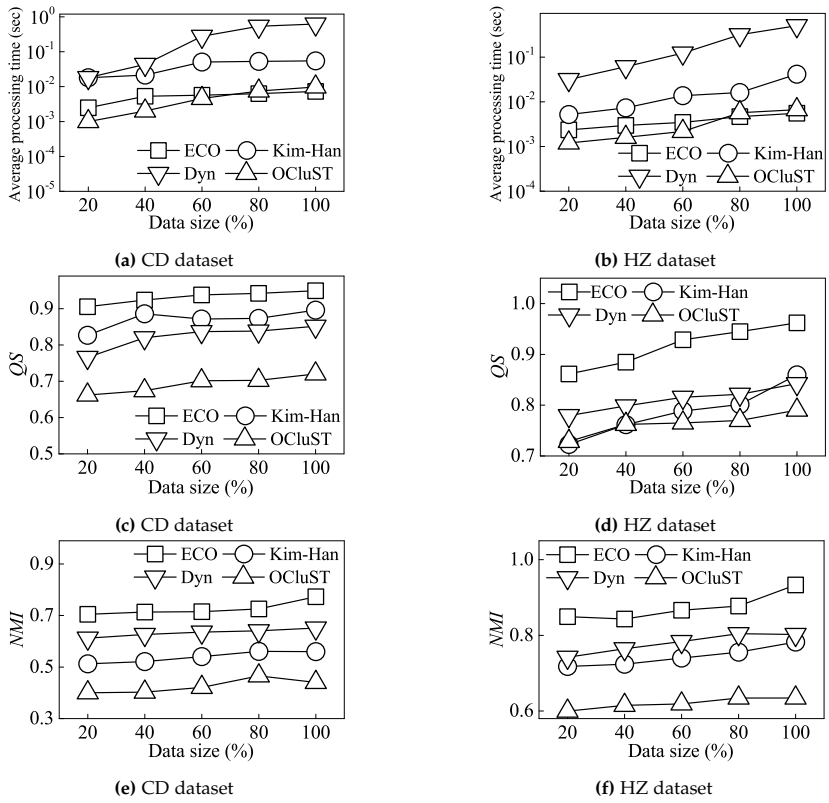


Figure C.9: Scalability

objects. Costa et al. [7] define a new metric for streaming trajectory clustering and process trajectories by means of non-separable Fourier transforms. Tra-POPTICS [9] explores the use of graphics processing units to improve efficiency. Similar to ECO, proposals in [38, 39] use an index structure, TC-tree, to facilitate efficient updates of clusters. Some studies maintain a micro-group structure, that stores compact summaries of trajectories to enable fast and flexible clustering [8, 22, 27, 29]. CUTis [8] continuously merges micro-groups into clusters, while CC_TRS [29], TCMM [22], and OCluST [27] generate macro groups on top of micro groups when requested by users. In the experimental study, we compare with the state-of-the-art streaming trajectory clustering method [27]. Some studies of real-time co-movement pattern mining also involve streaming trajectory clustering [5, 21, 32]. Comprehensive surveys of trajectory clustering are available [1, 40].

To the best of our knowledge, no existing studies of streaming trajectory clustering exploit temporal smoothness to improve clustering quality.

7.2 Evolutionary Clustering

Evolutionary clustering has been studied to discover evolving community structures in applications such as social [17] and financial networks [11] and recommender systems [4]. Most studies target k-means, agglomerative hierarchical, and spectural clustering [3, 6, 25, 26, 36]. Chakrabarti et al. [3] propose a generic framework for evolutionary clustering. Chi et al. [6] develop two functions for evaluating historical costs, PCQ (Preserving Cluster Quality) and PCM (Preserving Cluster Membership), to improve the stability of clustering. Xu et al. [36] estimate the optimal smoothing parameter α of evolutionary clustering. Recent studies model evolutionary clustering as a multi-objective problem [12, 23, 24, 37] and use genetic algorithms to solve it, which is too expensive for online scenarios. Dyn [37], the state-of-the-art proposal, features non-redundant random walk based population initialization and an improved particle swarm algorithm to enhance clustering quality.

The Kim-Han proposal [17] is the one that is closest to ECO. It uses neighbor-based smoothing and a cost embedding technique that smooths the similarity between each pair of nodes. However, ECO differs significantly from Kim-Han. First, the cost functions used are fundamentally different. Kim-Han's cost function is designed specifically for nodes in dynamic networks and is neither readily applicable to, or suitable for, trajectory data. In contrast, ECO's cost function is shaped according to the characteristics of the movements of trajectories. Second, Kim-Han smooths the similarity between each pair of neighboring nodes; in contrast, ECO smooths only the locations of a trajectory with abrupt movements, and the smoothing is performed only according to its most smoothly moving neighbor. Finally, Kim-Han includes iterative processes that degrade its efficiency, while ECO achieves $O(|\mathcal{O}_k|^2)$ complexity at each time step in the worst case and adopts a grid index to improve efficiency.

8 Conclusion and Future Work

We propose a new framework for evolutionary clustering of streaming trajectories that targets faster and better clustering. Following existing studies, we propose so-called snapshot and historical costs for trajectories, and formalize the problem of evolutionary clustering of streaming trajectories, called ECO. Then, we formulate ECO as an optimization problem and prove that it can be solved approximately in linear time, which eliminates the iterative processes employed in previous proposals and improves efficiency significantly. Further, we propose a minimal group structure and a seed point shifting strategy that facilitate temporal smoothing. We also present the algorithms necessary to enable evolutionary clustering along with a set of optimization techniques

that aim to enhance performance. Extensive experiments with two real-life datasets show that ECO outperforms existing state-of-the-art proposals in terms of clustering quality and running time efficiency.

In future research, it is of interest to deploy ECO on a distributed platform and to exploit more information for smoothing such as road conditions and driver preferences.

References

[1] J. Bian, D. Tian, Y. Tang, and D. Tao, "A survey on trajectory clustering analysis," *arXiv preprint arXiv:1802.06971*, 2018.

[2] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander, "Lof: identifying density-based local outliers," in *SIGMOD*, 2000, pp. 93–104.

[3] D. Chakrabarti, R. Kumar, and A. Tomkins, "Evolutionary clustering," in *SIGKDD*, 2006, pp. 554–560.

[4] J. Chen, C. Zhao, L. Chen *et al.*, "Collaborative filtering recommendation algorithm based on user correlation and evolutionary clustering," *Complex Syst.*, vol. 6, no. 1, pp. 147–156, 2020.

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

[6] Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng, "Evolutionary spectral clustering by incorporating temporal smoothness," in *SIGKDD*, 2007, pp. 153–162.

[7] G. Costa, G. Manco, and E. Masciari, "Dealing with trajectory streams by clustering and mathematical transforms," *Int. J. Intell. Syst.*, vol. 42, no. 1, pp. 155–177, 2014.

[8] T. L. C. Da Silva, K. Zeitouni, and J. A. de Macêdo, "Online clustering of trajectory data stream," in *MDM*, vol. 1. IEEE, 2016, pp. 112–121.

[9] Z. Deng, Y. Hu, M. Zhu, X. Huang, and B. Du, "A scalable and fast optics for clustering trajectory big data," *Cluster Comput*, vol. 18, no. 2, pp. 549–562, 2015.

[10] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise," in *SIGKDD*, vol. 96, no. 34, 1996, pp. 226–231.

[11] D. J. Fenn, M. A. Porter, M. McDonald, S. Williams, N. F. Johnson, and N. S. Jones, "Dynamic communities in multichannel data: An application to the foreign exchange market during the 2007–2008 credit crisis," *J Nonlinear Sci*, vol. 19, no. 3, p. 033119, 2009.

[12] F. Folino and C. Pizzuti, "An evolutionary multiobjective approach for community discovery in dynamic networks," *TKDE*, vol. 26, no. 8, pp. 1838–1852, 2013.

[13] J. Gan and Y. Tao, "Dynamic density based clustering," in *SIGMOD*, 2017, pp. 1493–1507.

[14] M. Gupta, C. C. Aggarwal, J. Han, and Y. Sun, "Evolutionary clustering and analysis of bibliographic networks," in *ASONAM*. IEEE, 2011, pp. 63–70.

[15] A. Idrissov and M. A. Nascimento, "A trajectory cleaning framework for trajectory clustering," in *MDC workshop*, 2012, pp. 18–19.

[16] C. S. Jensen, D. Lin, and B. C. Ooi, "Continuous clustering of moving objects," *TKDE*, vol. 19, no. 9, pp. 1161–1174, 2007.

[17] M.-S. Kim and J. Han, "A particle-and-density based evolutionary clustering method for dynamic networks," *PVLDB*, vol. 2, no. 1, pp. 622–633, 2009.

[18] L. Li, X. Chen, Q. Liu, and Z. Bao, "A data-driven approach for gps trajectory data cleaning," in *DASFAA*. Springer, 2020, pp. 3–19.

[19] T. Li, L. Chen, C. S. Jensen, and T. B. Pedersen, "Trace: real-time compression of streaming trajectories in road networks," *PVLDB*, vol. 14, no. 7, pp. 1175–1187, 2021.

[20] T. Li, R. Huang, L. Chen, C. S. Jensen, and T. B. Pedersen, "Compression of uncertain trajectories in road networks," *PVLDB*, vol. 13, no. 7, pp. 1050–1063, 2020.

[21] X. Li, V. Ceikute, C. S. Jensen, and K.-L. Tan, "Effective online group discovery in trajectory databases," *TKDE*, vol. 25, no. 12, pp. 2752–2766, 2012.

[22] Z. Li, J.-G. Lee, X. Li, and J. Han, "Incremental clustering for trajectories," in *DASFAA*. Springer, 2010, pp. 32–46.

[23] F. Liu, J. Wu, S. Xue, C. Zhou, J. Yang, and Q. Sheng, "Detecting the evolving community structure in dynamic social networks," *World Wide Web*, vol. 23, no. 2, pp. 715–733, 2020.

[24] F. Liu, J. Wu, C. Zhou, and J. Yang, "Evolutionary community detection in dynamic social networks," in *IJCNN*. IEEE, 2019, pp. 1–7.

[25] X. Ma and D. Dong, "Evolutionary nonnegative matrix factorization algorithms for community detection in dynamic networks," *TKDE*, vol. 29, no. 5, pp. 1045–1058, 2017.

[26] X. Ma, D. Li, S. Tan, and Z. Huang, "Detecting evolving communities in dynamic networks using graph regularized evolutionary nonnegative matrix factorization," *Physica A*, vol. 530, p. 121279, 2019.

[27] J. Mao, Q. Song, C. Jin, Z. Zhang, and A. Zhou, "Online clustering of streaming trajectories," *Front. Comput. Sci.*, vol. 12, no. 2, pp. 245–263, 2018.

[28] V. Patil, P. Singh, S. Parikh, and P. K. Atrey, "Geosclean: Secure cleaning of gps trajectory data using anomaly detection," in *MIPR*. IEEE, 2018, pp. 166–169.

[29] M. Riyadh, N. Mustapha, M. Sulaiman, N. B. M. Sharef *et al.*, "Cc_trs: Continuous clustering of trajectory stream data based on micro cluster life," *Math. Probl. Eng.*, vol. 2017, 2017.

[30] S. Song, C. Li, and X. Zhang, "Turn waste into wealth: On simultaneous clustering and cleaning over dirty data," in *SIGKDD*, 2015, pp. 1115–1124.

[31] A. Strehl and J. Ghosh, "Cluster ensembles—a knowledge reuse framework for combining multiple partitions," *J Mach Learn Res*, vol. 3, no. Dec, pp. 583–617, 2002.

[32] L.-A. Tang, Y. Zheng, J. Yuan, J. Han, A. Leung, C.-C. Hung, and W.-C. Peng, "On discovery of traveling companions from streaming trajectories," in *ICDE*. IEEE, 2012, pp. 186–197.

[33] J. Wang, P. Cheng, L. Zheng, C. Feng, L. Chen, X. Lin, and Z. Wang, "Demand-aware route planning for shared mobility services," *PVLDB*, vol. 13, no. 7, pp. 979–991, 2020.

[34] S. Wang, Y. Sun, C. Musco, and Z. Bao, "Public transport planning: When transit network connectivity meets commuting demand," in *SIGMOD*, 2021, pp. 1906–1919.

[35] H. Wu, C. Tu, W. Sun, B. Zheng, H. Su, and W. Wang, "Glue: a parameter-tuning-free map updating system," in *CIKM*, 2015, pp. 683–692.

[36] K. S. Xu, M. Kliger, and A. O. Hero III, "Adaptive evolutionary clustering," *Data Min Knowl Discov*, vol. 28, no. 2, pp. 304–336, 2014.

[37] Y. Yin, Y. Zhao, H. Li, and X. Dong, "Multi-objective evolutionary clustering for large-scale dynamic community detection," *Inf. Sci.*, vol. 549, pp. 269–287, 2021.

[38] Y. Yu, Q. Wang, and X. Wang, "Continuous clustering trajectory stream of moving objects," *China Commun.*, vol. 10, no. 9, pp. 120–129, 2013.

[39] Y. Yu, Q. Wang, X. Wang, H. Wang, and J. He, "Online clustering for trajectory data stream of moving objects," *Comput. Sci. Inf. Syst.*, vol. 10, no. 3, pp. 1293–1317, 2013.

[40] G. Yuan, P. Sun, J. Zhao, D. Li, and C. Wang, "A review of moving object trajectory clustering algorithms," *ARTIF INTELL REV*, vol. 47, no. 1, pp. 123–144, 2017.

[41] Y. Zeng, Y. Tong, and L. Chen, "Last-mile delivery made practical: An efficient route planning framework with theoretical guarantees," *PVLDB*, vol. 13, no. 3, pp. 320–333, 2019.

ISSN (online): 2446-1628
ISBN (online): 978-87-7573-983-7

AALBORG UNIVERSITY PRESS