MINING MOBILE OBJECT TRAJECTORIES: FRAMEWORKS AND ALGORITHMS

A Thesis
Presented to
The Academic Faculty

by

Binh Han

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in the
School of Computer Science, College of Computing

Georgia Institute of Technology
December 2013

MINING MOBILE OBJECT TRAJECTORIES: FRAMEWORKS AND ALGORITHMS

Approved by:

Professor Dr. Ling Liu,
Committee Chair
School of Computer Science, College of Computing
Georgia Institute of Technology

Professor Dr. Ling Liu, Advisor
School of Computer Science, College of Computing
Georgia Institute of Technology

Professor Dr. Edward Omiecinski,
Co-Advisor
School of Computer Science, College of Computing
Georgia Institute of Technology

Professor Dr. Calton Pu
School of Computer Science, College of Computing
Georgia Institute of Technology

Professor Dr. Shamkant Navathe
School of Computer Science, College of Computing
Georgia Institute of Technology

Dr. Gerald Lofstead
Scalable System Software Group
Sandia National Laboratories

Dr. Vikram Krishnamurthy
Nissan Research Center
Nissan Motor Company

Date Approved: November 11, 2013

To my family

and all those who have supported me through this journey.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor Dr. Ling Liu, without whose tremendous guidance, supervision and patience this dissertation could not have been possible. She has brought the best out in me and helped me grow. She has often believed in me more than I have believed in myself. I appreciate her for giving me the flexibility to pursue the problems I have found intriguing, while enthusiastically guiding me towards the right directions. I am grateful for the countless times she has spent on perfecting my work with great detail. Her relentless passion, amazing creativity and sharp vision when conducting research has been my constant source of inspiration. The invaluable lessons she has taught me both at work and in life will undoubtedly continue to shape my future.

I have been extremely fortunate to have Dr. Edward Omiecinski as my co-advisor. He has constantly encouraged me to explore new research topics. He has been willing to listen to any half-formulated ideas I conceived while carefully showing me how to analyze their feasibility. When I needed his advice, he has always made me a priority. His insightful advice has made our meetings fruitful. His encouragement has greatly helped boost my research productivity.

I wish to express my sincere gratitude to other members of my dissertation committee, Dr. Calton Pu, Dr. Shamkant Navathe, Dr. Gerald Lofstead and Dr. Vikram Krishnamurthy, whose constructive discussions have played a significant role in shaping my dissertation. It is a humbling experience to have such a distinguished team of researchers on board.

I would like to thank Dr. Leo Mark for being my faculty advisor in my first year at Georgia Tech and for introducing me to the Database Group.

I would like to acknowledge the Vietnam Education Foundation (VEF) for their financial support during my first two years at Georgia Tech and for providing career networking opportunities.

During my internships at PNNL and Amazon, I received indepth software development experience working on multiple interesting projects. Thanks to my internship mentors and colleagues, Jian Yin, Daniel Zhi, Saurabh Baji, Peter Sirota and Richard Cole, for helping me broaden my perspective and gain valuable work experience.

During my graduate studies, I have been lucky to be part of the DiSL research group that has provided me with a constant flow of stimulating intellectual challenges. This dissertation has benefited from related work of members in DiSL: Peter Pesti's development of the GTMobiSim mobility simulator, Bhuvan Bamba's work in location-based privacy, Myungcheol Doo's work in spatial indexing and Matt Weber's work in map-matching. The database lab with a lot of sunlight and kind officemates has made me feel at home. Many thanks to Danesh, Qingwang, Kisung, De, Kunal, Balaji, Rocky, Minh, Yuzhe and Qinyi, from whom I have learned a great deal.

My grad school years would have been a lot less colorful without the company of good friends inside and outside Atlanta. Special thanks go to Tien, Tien-Duy, Dung, Binh, Ly, Tung, Giang, Tan, Hang-Hung, Murtaza, Srikanth, Samantha, Al, Chris and Kevin for being supportive and generous. I am thankful to my best friends Huong-Cuong for always being there through the ups and downs.

Finally, this dissertation is dedicated to my parents, who have raised me to see life as fun and given me total freedom to be myself. My success is also theirs.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

SUMMARY

The proliferation of mobile devices and advances in geo-positioning technologies has fueled the growth of location-based applications, systems and services. Many location-based applications have now gained high popularity and have permeated the daily activities of mobile users. This has led to a huge amount of geo-location data generated on a daily basis, which draws significant interests in analyzing and mining ubiquitous location data, especially trajectories of mobile objects traveling in road networks (MO trajectories). Mobile trajectories are complex spatio-temporal sequences of location points with varying sample sizes and varying lengths. Mining interesting patterns from large collections of complex MO trajectories presents interesting challenges and opportunities which can reveal valuable insights into the study of human mobility in many perspectives. This dissertation research contributes original ideas and innovative techniques for mining complex trajectories from whole trajectories, from subtrajectories of significant characteristics, and from semantic location sequences within large-scale datasets of MO trajectories.

Concretely, the first unique contribution is the development of NEAT, a three-phase road-network aware trajectory clustering framework to organize MO subtrajectories into spatial clusters representing highly dense and highly continuous traffic flows in a road network. Compared with existing trajectory clustering approaches, NEAT yields highly accurate clustering results and runs orders of magnitude faster by smartly utilizing traffic locality with respect to physical constraints of the road network, traffic flows among consecutive road segments and flow-based density of mobile traffic as well as road network based distances. The second original contribution

of this dissertation is the design and development of TraceMob, a methodical and high performance framework for clustering whole trajectories of mobile objects. To our best knowledge, this is the first whole trajectory clustering system for MO trajectories in road networks. The core idea of TraceMob is to develop a road-network aware transformation algorithm that can map complex trajectories of varying lengths from a road network space into a multidimensional data space while preserving the relative distances between complex trajectories in the transformed metric space. The third novel contribution is the design and implementation of a fast and effective trajectory pattern mining algorithm TrajPod. TrajPod can extract the complete set of frequent trajectory patterns from large-scale trajectory datasets by utilizing space-efficient data structures and locality-aware spatial and temporal correlations for computational efficiency. A comprehensive performance study shows that TrajPod outperforms existing sequential pattern mining algorithms by an order of magnitude.

CHAPTER I

INTRODUCTION

The proliferation of mobile devices and advances in geo-positioning technologies have fueled the growth of location-based applications, systems and services. With the number of smartphones in use world wide reaching 1.4 billion units by the end of 2013 [1] and predictions of 2 billion units by 2015 [2], LBS revenue is forecasted to reach an annual global total of $13.5 billion by 2015 [3], up from $4 billion in 2012 [1]. Many location-based applications have now gained high popularity and have permeated the daily activities of mobile users, for example, Google Maps, Bing Maps, Google Latitude, Apple's FindMyFriends, FourSquare and FaceBook check-ins, to name a few. This has led to a huge amount of geo-location data generated on a daily basis, has drawn significant interest in analyzing and mining ubiquitous location data. We broadly classify location-based mining research into two categories based on the type of location data used as the processing unit for the analysis and mining of spatial datasets.

- The first class analyzes location data as position points and offers algorithms to query and mine point-based location data, for example, to find nearby points of interests or to discover hot-spot locations where people like to gather during weekends and holidays.

- The second class considers mobile object trajectories (MO trajectories) as units of location data processing and analysis. Trajectories typically represent moving paths of mobile objects in road networks or along walking paths. Trajectory mining aims at deriving new value and deep insights by analyzing and mining mobile object trajectories.

1.1 Motivation

This dissertation in particular focuses on the second class of location-based data mining, where we address the challenges and develop technical solutions for MO trajectory clustering and pattern mining. Given large-scale datasets of MO trajectories in road networks, many important questions need to be addressed, such as "Where are major traffic flows in the road network?", "What is the spatial trend in motion or movement behavior of mobile users?", "Where are the frequent places that many mobile trajectories are passing through during weekdays and weekends?", "Are there interesting trip patterns hidden from the large collection of mobile trajectories?". On one hand, these types of questions introduce exciting opportunities for mobile e-commerce and the mobile LBS industry, because answers to the questions can reveal valuable insights into the study of human mobility in many perspectives. Moreover, such knowledge significantly aids a wide range of location-based applications and services including traffic management, urban planning, geo-marketing and location-based recommendation systems. On the other hand, mining large-scale trajectories also poses interesting technical challenges.

The key challenges of MO trajectory analysis lie in the domain specific characteristics of MO trajectory data. First, MO trajectories are complex objects, consisting of temporal sequences of spatial location points. These sequences have varying sample sizes and represent moving paths of varying lengths. Second, trajectories of mobile users are typically constrained by the road network in which they travel since mobile objects can only move along road segments and make turns at road intersections. Third, the number of distinct geometric locations in a trajectory dataset is usually very large since there are hardly two identical geometric locations are recorded. In addition, trajectory datasets may contain a lot of outliers due to errors in location sensing and measurement collection. Thus, modeling trajectories and measuring their distances and spatio-temporal correlations to facilitate trajectory-based analysis and

mining remain open challenges, although distances and spatio-temporal correlations among point-based locations can be measured efficiently and accurately.

In this dissertation research, we aim at developing algorithms for efficient and accurate trajectory data modeling and trajectory based computation to analyze and mine MO trajectories by clustering whole trajectories, cluster subtrajectories of significant features and by sequential trajectory pattern mining. We approach the dual objectives for trajectory data modeling and trajectory-based computation with two design principles: It should capture the complex spatio-temporal characteristics of MO trajectories, and be simple enough to support efficient and high quality trajectory mining on a large scale in both online and offline scenarios.

1.2 Technical Challenges

In this dissertation, we aim to address the technical challenges in processing, analyzing and mining large-scale MO trajectory datasets from three types of trajectory data workloads: whole trajectories, subtrajectories of significant characteristics, and semantic location sequences within MO trajectories.

1.2.1 Analyzing and Clustering Whole Trajectories

We argue that a good representation of trajectories and a good metric to measure the distance between two trajectories are two critical components for efficient and effective clustering of whole trajectories and for deriving interesting and deep insights based on trip patterns of mobile users traveling on the roads.

Clustering is one of the most important data mining technique to discover the grouping structure in datasets. Clustering algorithms for traditional multidimensional data points have been studied extensively in the literature [13, 31, 38, 52, 53, 62, 67, 95, 98], including clustering location data points extracted from mobile object trajectories. However, there has been limited work on clustering full mobile object trajectories due to the domain specific characteristics of trajectory data. Unlike

multidimensional data points, which can be represented by fixed-size vectors for effective Euclidean distance measurements, trajectories are complex objects consisting of time-ordered sequences of spatial location points. These sequences have varying sizes, i.e., the number of recorded locations, and form road network paths of varying lengths. In addition, some trajectory datasets may contain a lot of outliers due to errors in data sensing and measurement collecting. MO trajectories may overlap with one another partially with respect to the road segments. However, some partially overlapped trajectories represent distinctly different trajectory clusters when grouping whole trajectories. Also, non-overlapping trajectories may be close to one another semantically or based on road network distance (e.g., mobile objects traveling on parallel roads), and thus, should belong to the same trajectory cluster. Finally, it is inefficient to perform clustering directly in the road network space where the centroid of a set of trajectories is hard and costly to compute. Interestingly, all these challenges reveal that the trajectory distance measure is a critical centerpiece for developing a high quality and high performance trajectory clustering algorithm. Although a number of distance functions have been proposed to measure trajectory similarities/dissimilarities, none of them are fully specialized for road-network MO trajectories. Existing distance measurements only consider Euclidean distance and examine all the recorded positions in a trajectory, while they might give acceptable performance for similarity trajectory querying (e.g., "find k nearest neighbours/most similar trajectories of a given trajectory"), they are not suitable for trajectory clustering where distance computations are performed within the scope of the whole dataset at once to discover the clustering structure.

1.2.2 Analyzing and Clustering Sub-trajectories

Many mobile objects may not travel the same route but they may still share similar parts of their whole trip. Clustering trajectories as a whole might miss interesting

groups of similar portions from MO trajectories. Therefore, clustering sub-trajectories is another important direction in trajectory clustering. In such component-based trajectory clustering schemes, trajectories are split into sub-trajectories which are then used as clustering units. As a result, a trajectory can have different portions belonging to different sub-trajectory clusters. However, existing approaches to sub-trajectory clustering are mainly based on density and Euclidean distance measures [11,36,56,58,59,74,91]. While previous approaches show reasonable performance for clustering trajectories of objects moving freely (e.g., the movement of animals through a forest or the movement of hurricanes across an ocean), they are inappropriate for clustering MO trajectories. We argue that when the utility of spatial clustering of mobile object trajectories is targeted at road-network aware location-based applications, density and Euclidean distance are no longer effective measures. In the context of a road network, traffic flows in the road network, the flow-based density characterization and the moving speeds of mobile objects become important factors which are needed to be captured effectively to find clusters of interesting sub-trajectories.

1.2.3 Mining Trajectory Patterns as Sequences of Semantic Locations

By clustering whole trajectories and sub-trajectories, we can reveal interesting motion and movement patterns of mobile users as collective groups. We argue that it is also interesting to analyze how trajectories share some subsequence of locations, or, what are the sequences of locations within the trajectory dataset, which are visited by many mobile users. Such trajectory sequence patterns for instance can reveal what types of mobile users are sharing the same or similar mobility patterns during their daily trips.

Sequential pattern mining has long been recognized as a primary mining task for analyzing sequential data, including purchase history, web logs and biological data.

The problem of sequential pattern mining has been studied extensively since it was first introduced by R. Agrawal and R. Srikant [10] in 1995 for *shopping basket* data, which are time-ordered list of shopping transaction event where each event is a set of items. Intuitively, existing sequential pattern mining algorithms can be applied to mine trajectory patterns since MO trajectories are temporal sequences of location points representing moving paths of mobile objects in road networks. However, since the granularity level of raw GPS locations is very low, an exact pattern of GPS locations hardly occurs multiple times in a trajectory dataset. Thus, the traditional notions of sequential patterns in transactional data cannot be applied directly to MO trajectory sequential pattern mining. Here, each event in a trajectory has only one item which is a road network location, as a user can only be at one location at a time, instead of multiple items per event as in traditional sequence data. Moreover, since MO trajectories are constrained by the road network as mobile objects can only move within road segments, there is a high degree of overlap in their temporal orders and spatial proximity. Therefore, general sequential pattern mining algorithms when applying to MO trajectories, without considering and utilizing their spatio-temporal characteristics, can suffer from exponential growths in space and computation.

The main challenges for efficient sequential pattern mining for MO trajectories include (1) defining a set of meaningful trajectory patterns as the desired mining output, (2) modeling MO trajectories with space-efficient data structures and (3) devising computational-efficient operation for tracking trajectory pattern frequency. The goal is to find the complete set of trajectory patterns, which helps discover interesting association rules between locations in large-scale MO trajectory datasets and can be applied in broad location-based applications such as trip recommendation, location prediction, location-based advertisement.

1.3 Dissertation Statement and Contributions

Before proceeding to the concrete contributions of this dissertation research, the dissertation statement can be formulated as follows:

Dissertation Statement: *This dissertation is the first one to provide methodical frameworks and configurable suites of algorithms for processing, clustering and mining mobile object trajectories in road networks, with high performance and high quality, by innovative utilization of complex spatial-temporal characteristics of mobile object trajectories, powered with novel data structures and computation models.*

This dissertation makes three unique contributions.

The first unique contribution of this dissertation is to develop NEAT, a three-phase road-network aware trajectory clustering framework for clustering mobile object sub-trajectories. Our method takes into account a number of road-network aware and motion aware metrics to organize MO sub-trajectories into interesting and meaningful spatial clusters, such as the traffic locality characterized by the physical constraints of the road network, the traffic flows among consecutive road segments, the flow-based density of traffic and shortest path distance. First, for a given set of MO trajectories, the road intersections can be viewed as the initial partitioning points where trajectories can be split into atomic sub-trajectories, called trajectory fragments. Second, the trajectory fragments corresponding to a road segment can be viewed as a locally dense cluster of objects involved in the given set of trajectories. Third, trajectory fragments should be clustered based on their continuity with regard to the traffic flows on the consecutive road segments. In addition, the proximity measure in a road network space uses shortest path distance instead of Euclidean distance. NEAT efficiently discovers clusters of sub-trajectories which describe both highly dense and highly continuous traffic flows of mobile objects in a road network.

The second original contribution of this dissertation is the design and development of the TraceMob framework for clustering full trajectories of mobile objects. TraceMob has three unique features. First, we introduce a grid-based technique to measure the distance between two mobile object trajectories of varying sizes. Second, we develop a multidimensional scaling algorithm to transform complex MO trajectories in a road network space into multidimensional points in a Euclidean space while preserving the relative distances of trajectories in the transformed metric space. This transformation enables us to perform k-means clustering effectively on the mapped multidimensional points of the trajectories. We introduce a grid-based cluster validation metric as an integral part of the TraceMob framework for measuring the quality of trajectory clusters. Extensive experiments show that TraceMob outperforms state of art approaches in terms of both robustness and effectiveness.

The third novel contribution of this dissertation is the TrajPod algorithm for trajectory pattern mining. We define trajectory patterns as time-ordered sequences of semantic spatial units and propose a fast locality-aware approach to extract the complete set of frequent trajectory patterns from a given MO trajectory dataset. We introduces a vertical layout of MO trajectories consisting of trajectory id-lists associated with the semantic spatial units, which are partitioned into locality-aware sublists. We also use a space-efficient representation for multiple occurrences of a semantic spatial unit in a trajectory with respect to each sublist. These data structures combined with locality-aware join operators allow our method to mine trajectory patterns efficiently with early candidate pruning and fast support counting. A comprehensive performance study shows that TrajPod outperforms existing sequential pattern mining algorithms by an order of magnitude.

1.4 Organization of the Dissertation

The rest of this dissertation will be organized as a series of chapters, each one dedicated to a specific topic within the scope of mobile object trajectory mining. Each chapter will be organized in the following format. The introduction section gives the problem background, motivation, guidelines and overview of our approach. The main section provides an in-depth presentation of our technical contributions. The experimental evaluation section demonstrates the efficiency and effectiveness of our approach. The related work section gives a review of relevant literature. The last section concludes the chapter. More specifically, Chapter 2 and Chapter 3 are dedicated to MO trajectory clustering. Chapter 2 presents NEAT, our three-phase road-network aware clustering framework which integrates locality, flow and density measures to discover spatial clusters of MO sub-trajectories representing higly dense and highly continuous traffic flows. Chapter 3 presents TraceMob, a methodical approach to clustering MO trajectories as a whole with a novel trajectory distance measure and robust trajectory-to-multidimentional point transformation for effective partitioning clustering. Chapter 4 is dedicated to trajectory pattern mining where we formulate our desired trajectory patterns and present TrajPod, a fast locality-aware MO trajectory pattern mining algorithm which achieves both space efficiency and computation efficiency. Chapter 5 presents the design of our TrajBox software toolkit for MO trajectory processing and mining to support the development of trajectory-based applications. Chapter 6 discusses future research directions and concludes the dissertation.

1.5 Bibliographic Notes

Chapter 2 contains material from publications co-authored with Ling Liu and Edward Omiecinski [40, 43]. Material in Chapter 3 appears in our paper [42]. Material in Chapter 4 appears in our paper [41]. The implementations described in the

dissertation are available at [4].

CHAPTER II

ROAD-NETWORK AWARE TRAJECTORY CLUSTERING: INTEGRATING LOCALITY, FLOW AND DENSITY

Mining trajectory data has been gaining significant interest in recent years. However, existing approaches to trajectory clustering are mainly based on density and Euclidean distance measures. We argue that when the utility of spatial clustering of mobile object trajectories is targeted at road-network aware location-based applications, density and Euclidean distance are no longer the effective measures. This is because traffic flows in a road network and the flow-based density characterization become important factors for finding interesting trajectory clusters. We propose NEAT−a road-network aware approach for fast and effective clustering of trajectories of mobile objects travelling in road networks. Our approach carefully considers the traffic *locality* characterized by the physical constraints of the road network, the traffic *flow* among consecutive road segments, and the flow-based *density* to organize trajectories into spatial clusters in a comprehensive three-phase clustering framework. NEAT discovers spatial clusters as groups of sub-trajectories which describe both dense and highly continuous flows of mobile objects. We perform extensive experiments with mobility traces generated using different scales of real road networks. Experimental results demonstrate the flexibility of the NEAT system and show that NEAT is highly accurate and runs orders of magnitude faster than existing density-based trajectory clustering approaches.

2.1 Introduction

Recent years have witnessed a rapidly growth in the field of location-based services (LBSs) and applications due to the pervasive use of GPS receivers and WiFi or location sensing technology embedded in mobile devices (e.g., cellular phones, automobiles). With the number of smartphones in use world wide reached 1.038 billion units in 2012 and is predicted to reach 2 billion units by 2015 [2], LBS revenue is forecasted to reach an annual global total of $13.5 billion by 2015 [3], up from $4 billion in 2012 [1]. Ubiquitous GPS/WiFi-enabled mobile devices generate a huge amount of trajectory data, which are sequences of time-ordered locations of mobile objects. There has been a lot of work on collecting, storing, indexing and querying trajectories of mobile objects [47] [22] [18] [28] [11, 46, 71]. We refer to the trajectories of mobile objects in a road network as *MO trajectories*. Clustering trajectories of these objects provides the most value and has a wide range of LBS applications. For example, the resulting clusters would help provide knowledge about traffic flows as well as dense areas in a road network. Such knowledge is very useful for applications in vehicular ad hoc network (VANET) [49] [72], traffic monitoring [63], transportation planning [51] and location-based advertising [35, 82, 92]. We briefly present below two interesting application scenarios which show the usefulness of trajectory clustering and motivate us to study the problem of clustering trajectories of mobile objects moving in a spatially constrained road network.

Public transit planning. The establishment of public transportation systems always target the road network routes that can maximize the utilization of public transportation vehicles. Since a bus runs along consecutive road segments, in order to make the best use of the bus service regarding the number of passengers it can serve and to improve public transport convenience, e.g., reduce the number of transit stops for passengers, knowing which routes with not only high density but also high continuity helps optimize rail/bus line and terminal arrangement.

Location based advertising on mobile devices. Mobile advertisers are trying to improve the matching of user locations and marketing information. It would be beneficial for local stores to send advertisements, e.g., special offers or discounts, to mobile devices taking path in major traffic flows passing by their stores. For example, consider when a store wants to send out text messages with a discount coupon. If the text message is sent to a group of people in a nearby dense road segment which is also part of a route with significant traffic flow leading to the store, it will better increase the chance that people receiving the coupon will actually come to the store during their trips, compared to when the message is sent to a dense area but does not belong to the traffic flow passing by their store.

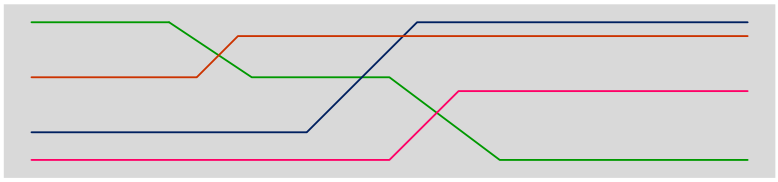
A straight forward solution to address this clustering problem is to adapt the traditional density-based clustering algorithms (e.g., DBSCAN [31] or OPTICS [13] - a variant of DBSCAN) to group similar *MO* trajectories. However, clustering trajectories as a whole does not take into account similar sub-trajectories since trajectories have various lengths. This is addressed by partial trajectory clustering. The TraClus algorithm [36] is the representative method for clustering portions of a trajectory instead of the whole trajectory. Specifically, TraClus is a two phase clustering algorithm. In the partitioning phase, each trajectory is first examined sample by sample to identify a sequence of characteristic points at which the moving object makes a sharp turn, called a rapid change in direction, and then the trajectory is partitioned into line segments by these characteristic points. The grouping phase performs a DBSCAN style clustering on those line segments to find similar sub-trajectories. A drawback to this and most similar partial trajectory clustering approaches is that they only consider distances in Euclidean space. While they show reasonable performance for clustering trajectories of objects moving freely (e.g., the movement of animals through a forest or the movement of hurricanes across an ocean), they are inappropriate for clustering *MO* trajectories. We argue that clustering *MO* trajectories

should take into account the traffic locality characterized by the spatial constraints of the underlying network, e.g., road segments and intersections, the movement behavior of mobile objects as well as the traffic flows on consecutive road segments, in addition to mobile object density and the road network proximity. We illustrate our positional statements with the following set of examples.

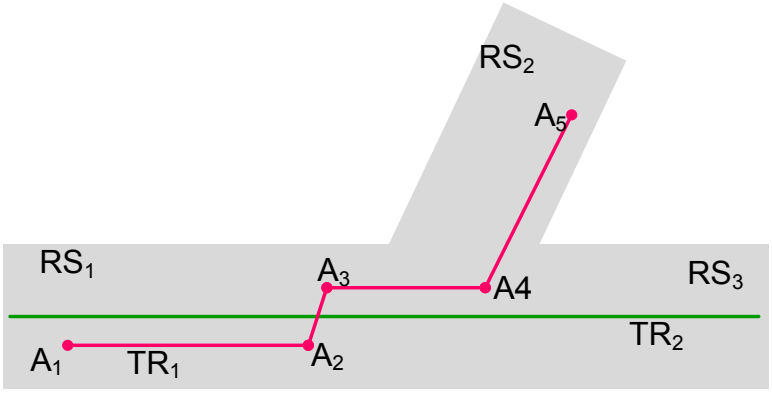
Example 1 We have a set of four objects moving along the same road segment and produce four *MO* trajectories as shown in Figure 1(a). In the context of a road network, those objects have similar movement behavior with respect to the road segment. Therefore, their trajectories should be grouped together regardless of the difference in their specific movement on the road segment. Hence, it is unnecessary to further partition these trajectories, even though some rapid changes in direction are found in those trajectories.

Example 2 Consider two trajectories TR_1 and TR_2 in Figure 1(b). TR_1 describes an object moving on road segment RS_1, changing lanes, and then making a left turn to road segment RS_2. TR_2 describes another object moving straight from RS_1 to RS_3. If we use the TraClus algorithm, to cluster the *MO* trajectories, TR_1 will be partitioned into four line segments: A_1A_2, A_2A_3, A_3A_4, and A_4A_5, while TR_2 consists of only one line segment that is too long to be grouped with the segments of TR_1 on RS_1 (A_1A_2, A_2A_3, and A_3A_4). We argue that although there is no significant turning point found in TR_2, we should still partition it into two line segments fragments corresponding to the two distinct road segments RS_1 and RS_3.

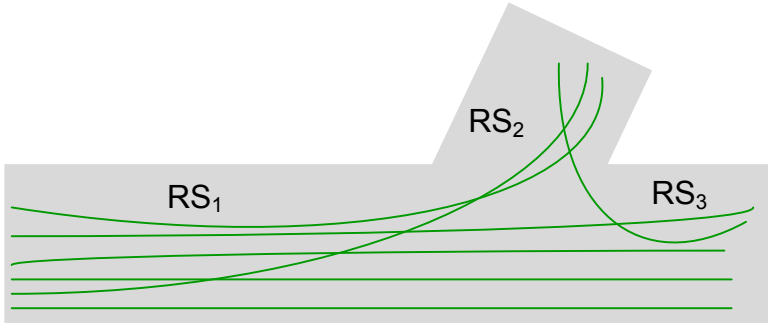
Example 3 In Figure 1(c), road segments RS_1 and RS_3 share a larger number of mobile objects than that of road segments RS_1 and RS_2. Thus, when consider traffic continuity, traffic on road segment RS_1 should be merged with traffic on RS_3 rather than RS_2.



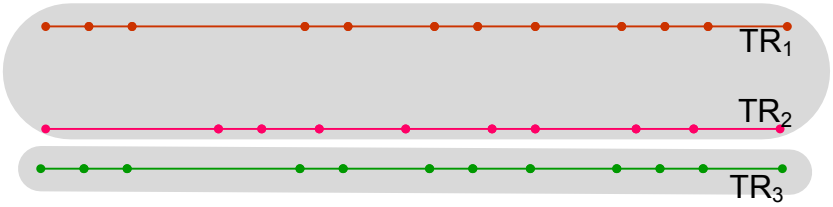
(a) An example of the movement of four vehicles on a road segment



(b) Parts of two trajectories on the same road segment that are overlooked in TraClus



(c) Traffic on road segment RS_1 should be merged with traffic on road segment RS_3 rather than RS_2



(d) An example showing the difference between the Euclidean proximity and the road network proximity

Figure 1: Illustrative examples

Example 4 We have three trajectories: TR_1 and TR_2 are on the same road segment, and TR_3 is on another road segment as shown in Figure 1(d). Using the Euclidean

distance, TR_3 is closer to TR_2 than TR_1, even though using the road network distance (either segment length based or travel time based), TR_1 is closer to TR_2 than TR_3.

In this chapter, we present NEAT−a road NEtwork Aware approach to Trajectory clustering. To the best of our knowledge, NEAT is the first technique to address the *MO* trajectory clustering problem by taking into account the continuity of movements restricted by the underlying road network, the network proximity and the traffic flows among consecutive road segments to organize *MO* trajectories into spatial clusters. The clusters discovered by NEAT are groups of sub-trajectories, which describe both dense and highly continuous traffic flows of mobile objects. A unique feature of the NEAT framework is the formulation of the three phase trajectory partitioning, merging and clustering process. Specifically, based on the observations from the above examples, we identify three important design guidelines. First, for a given set of *MO* trajectories, the road intersections can be viewed as the initial partitioning points where trajectories can be split into atomic sub-trajectories called trajectory fragments. Second, the trajectory fragments corresponding to a road segment can be viewed as a locally dense cluster of objects involved in the given set of trajectories. Third, trajectory fragments should be clustered based on their continuity with regard to the traffic flows on the consecutive road segments. In addition, the proximity measure in a road network space uses shortest path distance instead of Euclidean distance. We perform extensive experiments with road network mobility traces generated using different scales of road network maps. Our experimental results demonstrate that the NEAT approach is highly efficient and accurate. It can run more than three orders of magnitude faster than existing density-based trajectory clustering approaches.

The rest of this chapter is organized as follows. Section 2.2 presents the reference road network model and an overview of our NEAT computational model and framework architecture. Section 2.3 describes the algorithms used in each of the three phases of the NEAT framework. We report our experimental results in Section 2.4,

discuss related work in Section 2.5 and conclude the chapter in Section 2.6.

2.2 NEAT model and framework

We first describe a reference model for road networks and present the basic concepts and operations of the NEAT model. We end this section with a brief overview of the NEAT three phase framework and an illustrative example.

2.2.1 Road-Network Model

A road network is represented by a single directed graph $G = (\mathcal{V}, \mathcal{E})$, composed of the junction nodes $\mathcal{V} = \{n_0, n_1, \ldots, n_N\}$ and directed edges $\mathcal{E} = \{(sid, n_i n_j) | n_i, n_j \in \mathcal{V}\}$.

An edge $e = (sid, n_i n_j) \in \mathcal{E}$ representing a road segment connecting two junctions n_i and n_j in the real road network. The listed order $n_i n_j$ indicates the direction from n_i to n_j of the road segment. For road segments which have bidirectional lanes, we use edge $e = (sid, n_i n_j)$ and $e' = (sid, n_j n_i)$ to denote the fact that the road segment is bi-directional and we label each edge with the corresponding road segment identifier *sid*. The length of a road segment $e = (sid, n_i n_j)$ is denoted by $length(n_i n_j)$.

Let $L(e)$ denote the set of adjacent edges of $e = (sid, n_i n_j)$ and $L_{n_i}(e)$ denote the set of adjacent edges of e, which connect to e at junction n_i. Hence, we have $L(e) = L_{n_i}(e) \cup L_{n_j}(e)$. If n_i is a dead-end node connected by edge e, then $L_{n_i}(e) = \phi$. If two edges e_i and e_j are adjacent, function $I(e_i, e_j)$ will return the junction node (intersection) of these two edges. A route in the road network G is a network path $e_0 e_1 ... e_k$ such that $e_{i+1} \in L(e_i)$ $(0 \leq i < k)$.

We define a *road network location* of a mobile object as a tuple of three elements: sid − the identifier of the road segment on which this object resides, the geometric coordinates (x, y) of the position of the object on the road segment *sid*, and the timestamp t when the position is recorded, denoted by $l = (sid, x, y, t)$. A road network location can also be represented by a tuple (sid, p, t) where p is the offset of the location from the start junction of the road segment identified by *sid*. We use the (x, y)

coordinates to represent location in this chapter due to the popularity of geometric coordinates. We measure the network proximity between two network locations using the network-based distance metric [95]. Let l_i and l_j denote the network locations that belong to the edge $e_i = (sid_i, n_a n_b)$ and the edge $e_j = (sid_j, n_c n_d)$ respectively and $e_i \neq e_j$. The network distance between l_i and l_j is the length of the shortest path between l_i and l_j, denoted by $dist_N(l_i, l_j)$. We use the terms point and location interchangeably to refer to a road network location and the terms junction, intersection and endpoint interchangeably to refer to a road junction. We use $(sid, n_i n_j)$ and $n_i n_j$ interchangeably to denote a segment connecting two end nodes n_i and n_j when there is no confusion.

2.2.2 NEAT model

In this section, we define the basic concepts and operations of our road network aware trajectory model with illustrative examples.

For each mobile object, each of his/her trips with a beginning location and a destination location forms a trajectory. A *trajectory*, denoted by $TR = (trid, l_0 l_1 ... l_n)$, is a time-ordered sequence of locations $l_0, l_1, ..., l_n$ of an MO in the road network over time and uniquely identified by a trajectory identifier $trid$. A subsequence of points in a trajectory forms a *sub-trajectory*. Note that in our model, the temporal information in a trajectory, i.e., the recorded timestamps, determines the order of locations in the trajectory. Therefore, the direction of movement of an object is always preserved. For presentation convenience, we do not explicitly mention the directions of movement in the definitions and figures used in the subsequent sections of the chapter when no confusion occurs.

Definition 1 *(t-fragment)* Let $TR = \{trid, l_0 l_1 ... l_n\}$ denote a trajectory consisting of $n+1$ points and $trid$ denote the trajectory identifier. A t-fragment of TR, denoted by $tf = \{trid, sid, l_k l_{k+m}\}$, represents a sub-trajectory $l_k l_{k+1} ... l_{k+m}$ consisting of $m+1$

consecutive points extracted from TR which lie on the same road segment sid, i.e., $l_i.sid = l_j.sid$ $(\forall i, j : k \leq i, j \leq k + m, i \neq j)$.

There can be more than one t-fragment associated with a road segment for a given trajectory (in case the mobile object travels on the road segment multiple times during her trip).

Definition 2 *(base cluster)* Let $\mathcal{T}$ denote a set of trajectories and e denote a road segment. A *base cluster* S with respect to e is a group of distinct t-fragments, each of these t-fragments belongs to a trajectory in $\mathcal{T}$ and is associated with e. The *base cluster* S is formally defined as follows:

$S = \{tf_i | TR(tf_i) \in \mathcal{T}, tf_i.sid = e.sid\}$ where $TR(tf_i)$ denotes the trajectory from which the t-fragment $tf_i \in S$ is extracted. The road segment e is called the representative of the base cluster S, and is denoted by e^S. The base cluster S is said to be associated with the road segment e^S.

We call a trajectory that has t-fragments in a base cluster the *participating trajectory* of the base cluster.

Definition 3 *(trajectory cardinality)* The set of participating trajectories of a base cluster S is defined as: $PTr(S) = \{TR(tf_i) | \forall tf_i \in S\}$. The cardinality of $PTr(S)$, denoted by $|PTr(S)|$, is called the *trajectory cardinality* of S.

Definition 4 *(cluster density)* The *density of a base cluster* S, denoted by $d(S)$, is the number of t-fragments in S. Given $B = \{S_0, S_1, ..., S_N\}$ is a set of base clusters, we call the base cluster with the highest density in B the *dense-core* of B, denoted by $densecore(B)$.

Definition 5 *(netflow)* The *netflow* between two base clusters S_i and S_j, denoted by $f(S_i, S_j)$, is the number of trajectories participating in both clusters: $f(S_i, S_j) = |PTr(S_i) \cap PTr(S_j)|$

The function *netflow* between two base clusters computes the number of common objects traveled on both representative road segments e^{S_i} and e^{S_j}.

Definition 6 *(f-neighborhood)* Let B denote a set of base clusters, S_i denote a base cluster and n_u denote one endpoint of e^{S_i}. The f-neighborhood of S_i with respect to n_u, denoted by $N_f(S_i, n_u)$, is the set of base clusters that have at least one common participating trajectory, and is formally defined as: $N_f(S_i, n_u) = \{S_j |\ e^{S_j} \in L_{n_u}(e^{S_i})\ \&\ f(S_i, S_j) > 0\}$.

Let n_v be the other endpoint of e^{S_i}. We define the f-neighborhood of S_i with respect to e^{S_i} as: $N_f(S_i) = N_f(S_i, n_u) \cup N_f(S_i, n_v)$. Each $S_j \in N_f(S_i)$ is called the f-neighbor of S_i. Note that the f-neighbor is a symmetric relation.

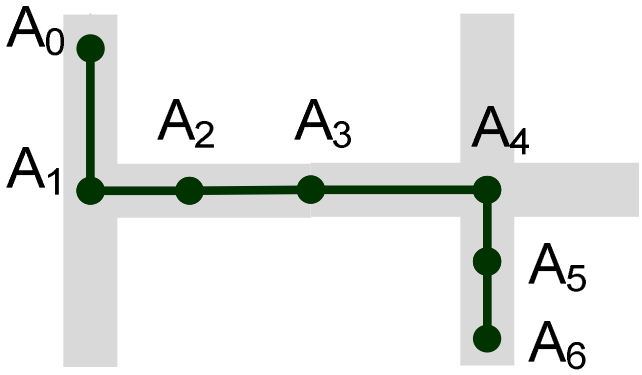
Definition 7 *(maxFlow-neighbor)* Let S_i denote a base cluster and n_u denote one endpoint of e^{S_i}. We call S_k the $maxFlow$-neighbor of S_i at n_u, denoted by $maxFlow(S_i, n_u)$, if $f(S_i, S_k) = max\{f(S_i, S_j) |\ S_j \in N_f(S_i, n_u)\}$. $f(S_i, S_k)$ is called a $maxFlow$ of S_i.

Definition 8 *(flow cluster)* A *flow cluster* (or a *flow* for short) is an ordered list of base clusters, denoted by $F = \{S_0, S_1, ..., S_N\}$, where $S_{i+1} \in N_f(S_i)(0 \leq i < N)$ and $e^{S_0}e^{S_1}...e^{S_N}$ forms a route in the road network. We call $e^{S_0}e^{S_1}...e^{S_N}$ the representative route of F, denoted by r_F.

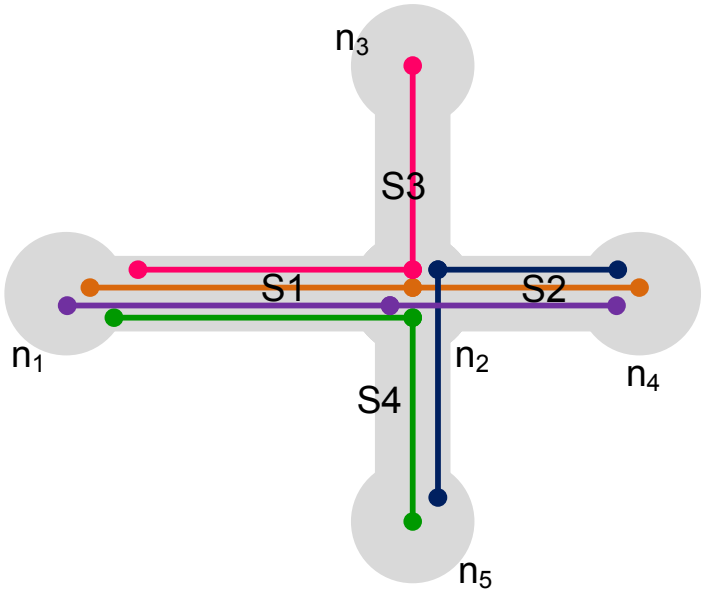
Since a base cluster is comprised of t-fragments and a flow cluster is comprised of base clusters, we say that a flow cluster is comprised of t-fragments. Therefore the definition of *trajectory cardinality* also applies to a flow cluster. We define the netflow between a flow cluster F and a base cluster S as $f(F, S) = |PTr(F) \cap PTr(S)|$.

Figure 2(a) shows a trajectory that can be represented by a sequence of three t-fragments A_0A_1, A_1A_4 and A_4A_6. In Figure 2(b), we have five trajectories located on four road segments n_1n_2, n_2n_3, n_2n_4 and n_2n_5. There are three t-fragments which lie on n_1n_2. Those three t-fragments are grouped together in base cluster S_1 whose

representative road segment is n_1n_2. In total, we have a set of base clusters $B = \{S_1, S_2, S_3, S_4\}$. The density of S_1 is $d(S_1) = 4$. Similarly, we have $d(S_2) = 3$, $d(S_3) = 1$ and $d(S_4) = 2$. S_1 is the dense-core of B since $d(S_1) = 4$ is the highest density. The netflows among these base clusters are: $f(S_1, S_2) = 2$, $f(S_1, S_3) = 1$, $f(S_1, S_4) = 1$, $f(S_2, S_3) = 0$ and $f(S_2, S_4) = 1$. The f-neighborhood of S_1 with respect to n_2 is $N_f(S_1, n_2) = \{S_2, S_3, S_4\}$, in which S_2 is the $maxFlow$-neighbor of S_1. The possible flow clusters include $\{S_1, S_2\}$, $\{S_1, S_3\}$, $\{S_1, S_4\}$ and $\{S_2, S_4\}$.



(a) A trajectory has three t-fragments



(b) An example of base clusters and flow cluster

Figure 2: Examples of the elements in the NEAT model

2.2.3 NEAT Framework Overview

Given a road network $G = (\mathcal{V}, \mathcal{E})$ and a set of N trajectories collected from mobile objects travelling on G, denoted by $\mathcal{T} = \{TR_1, TR_2, ..., TR_N\}$, NEAT performs road network aware trajectory clustering in three phases:

Phase 1 - Base cluster formation: We transform the given set of *MO* trajectories into a set of trajectory fragments (t-fragments). Then we organize these t-fragments into *base clusters* by grouping those t-fragments that correspond to the same road segment into one base cluster.

Phase 2 - Flow cluster formation: We selectively merge *base clusters* into *flow clusters* based on the major mobility flows and the flow continuity inherent in the given set of trajectories.

Phase 3 - Flow cluster refinement: We optimize the clustering result using a density-based refinement method. Our density-based optimization modifies the widely-used Hausdorff distance [16] with the shortest path measurement and adapts the DBSCAN clustering algorithm [31].

The final result produced by NEAT is a partitioning of the given *MO* trajectories into a set of trajectory clusters $\mathcal{O} = \{C_1, C_2, ..., C_K\}$ where each cluster $C_i (0 \leq i \leq K)$ contains a set of trajectory fragments satisfying two criteria:

- *High density*: the trajectory fragments in the same cluster are within the network proximity of each other.

- *High continuity*: the trajectory fragments in the same cluster show a major traffic flow in the given trajectory data.

The NEAT system uses 3-tier client/server architecture. Each client node acts as a mobile device which records its locations, sends its trajectories to a NEAT server and makes requests to the server to get trajectory clustering results for a particular road network. NEAT server also distributes trajectory datasets across multiple nodes in a

cluster. These data nodes can perform some data preprocessing tasks. In this chapter, we focus on the trajectory clustering application running on the NEAT server.

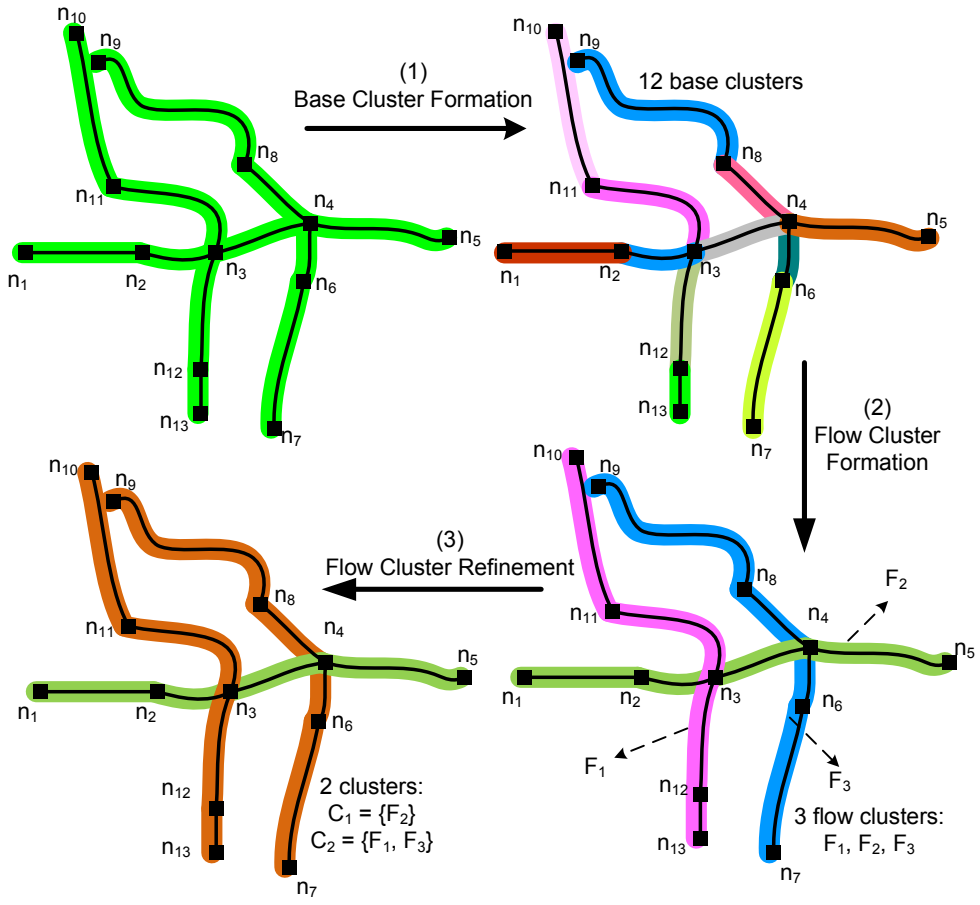


Figure 3: An example of three phase clustering in the NEAT framework.

Figure 3 illustrates the three phase NEAT framework. Consider a set of trajectories located on 12 road segments in an example road network as shown in the upper-left of Figure 3. In Phase 1, a set of base clusters are constructed from the given set of trajectories (the upper-right of Figure 3). In Phase 2, we utilize road network information and mobile object movement characteristics to group base clusters into flow clusters. Important operations include computing the *netflow*s, finding the f-neighborhoods and compute the $maxFlow$-neighbors. For our example, the result of this phase is a set of three flow clusters $\{F_1, F_2, F_3\}$ (the bottom-right of Figure 3). In Phase 3, we refine the resulting flow clusters by merging those flow clusters that

are close in terms of a network based distance measure and a distance threshold. In Figure 3, F_1 and F_3 are merged in Phase 3 to form a larger trajectory cluster. The two clusters C_1 and C_2 shown in the bottom-left of Figure 3 are the final result of clustering for this specific example.

2.3 Three phase trajectory clustering

In this section, we present in details the algorithms used in the NEAT clustering framework to produce base clusters in Phase 1, flow clusters in Phase 2 and the final trajectory clusters in Phase 3.

2.3.1 Phase 1 - Base Cluster Formation

We perform the base cluster formation in two steps. First, we examine the input set of *MO* trajectories and partition each *MO* trajectory into a sequence of t-fragments. Second, we group those t-fragments that belong to the same road segments into one *base cluster*.

2.3.1.1 Partitioning Trajectories into t-fragments

Since a mobile object moves along contiguous road segments, two consecutive locations recorded in a *MO* trajectory are either on the same road segment or on two different road segments. In the latter case, the two different road segments are either contiguous or lie on the route (travel path) of the mobile object such that they are connected through a sequence of road junction nodes on the same path. For each trajectory $TR_k = \{trid_k, l_0 l_1 ... l_n\}$ in the given trajectory dataset, we start the t-fragment extraction by examining TR_k from the first location l_0 to the last location l_n in the sequence of location samples $\{l_0 l_1 ... l_n\}$ of the trajectory. Based on the fact that a mobile object has to go through the road intersection when moving between two contiguous road segments, we take every two consecutive points in the trajectory, say l_i and l_{i+1}, and check if their road segment identifiers, denoted by $sid(l_i)$ and $sid(l_{i+1})$

respectively, are different. If $sid(l_i) \neq sid(l_{i+1})$, we know that l_i and l_{i+1} are on different road segments. If they are contiguous, we can obtain the road junction node that intersects these two road segments. If the two road segments happen to be not contiguous, we can obtain the sequence of road junction nodes connecting them on the travel path of the object using the map-matching approach [89]. Next, we insert the obtained junction node(s) as new points in between l_i and l_{i+1} in the trajectory being examined. The junction nodes added to a trajectory in this phase are marked as different points than the original location samples. After examining every point in a given trajectory TR_k, the sequence of junction nodes added to TR_k will serve as the trajectory splitting points used to partition the trajectory into t-fragments. When a set of trajectories are given as time series of geometric coordinates, NEAT will first preprocess the set of trajectories using map-matching (MM) algorithms such that each point in a trajectory is mapped to a road network location as defined in Section 2.2.1. We use the SLAMM map-matching algorithm [89] in this data prepocessing step. MM algorithms for bulk location data are more effective as noted in [89] because look-ahead and look-around algorithms can catch many known errors of earlier MM algorithms, such as map-matching location samples between two nearby parallel road segments.

By transforming a trajectory into a set of t-fragments, only the first point and the last point in the original trajectory are kept, together with the newly inserted road junction points. These points play critical roles in extracting t-fragments and constructing base clusters in the next step of Phase 1 as well as in subsequent phases of NEAT. Furthermore, the sequence of t-fragments extracted from a trajectory still maintains the traveling route, the movement direction as well as the identifier of the original trajectory.

2.3.1.2 Grouping t-fragments into Base Clusters

We examine the t-fragments extracted from the MO trajectories and group them by their road segment identifiers. Each group of t-fragments corresponding to a road segment forms one base cluster with the road segment as its representative (Definition 2). As discussed in Section 2.1, the t-fragments on the same road segment are considered close in terms of network proximity and they display similarity in the movement of their mobile objects. We compute the density of the resulting base clusters (Definition 4), then sort them by their densities in descending order. The output of Phase 1 is a sorted list of base clusters with the first base cluster as the dense-core of the set of base clusters. The base clusters are used as the building blocks of our flow-based clustering in the next phase. We will use flow and density controlled merging algorithms to construct the final trajectory clusters of a given trajectory dataset $\mathcal{T}$.

2.3.2 Phase 2 - Flow Cluster Formation

The flow-based clustering algorithm takes as an input the list of base clusters B produced from Phase 1. It starts by selecting one base cluster in B as the first initial flow cluster. It then expands this initial cluster by adding other base clusters one at a time such that the representative road segments of the base clusters selected for merging are concatenated to make a route. This expanding process will stop when every base cluster in B has been examined for its potential to be merged with existing flow clusters. We consider flow, density and road speed limit as three characteristics of a traffic stream to define a set of merging criteria. We construct flow clusters by grouping base clusters according to these criteria. We discuss below how to choose the initial base cluster and determine which base clusters are the best candidates for merging with the flow cluster under consideration.

2.3.2.1 Density-based Flow Cluster Initialization

In the first prototype of NEAT, we take the dense-core of the density-ordered list of base clusters B to begin the flow-based clustering process. There are at least two reasons for choosing $densecore(B)$ as the initial flow cluster to start Phase 2. Given that a major traffic stream usually covers the road segment(s) with the highest traffic density in the road network, if we randomly pick a base cluster in B to initialize a flow cluster, it might lead to a flow cluster that describes a negligible traffic stream and will eventually be filtered out. In addition, choosing $densecore(B)$ also guarantees that the set of base clusters are merged in a deterministic order. Hence, the resulting flow clusters are always the same for the same input set of trajectories.

2.3.2.2 f-neighbor Merging for Flow Clusters

In this section, we describe how to merge a base cluster into an existing flow cluster. Suppose we are in the process of expanding a flow cluster F, which is in the form of an ordered list of base clusters, denoted by $\{S_a S_{a+1} \dots S_{b-1} S_b\}$. We extend the list by inserting a base cluster either at the front or at the end of the list. Inserting a base cluster at the front of the list is performed similarly to inserting a base cluster at the end of the list. Let n_u denote the other end point of e^{S_b} other than the intersection $I(e^{S_{b-1}}, e^{S_b})$ of the two representative road segments of base clusters S_{b-1} and S_b. The base cluster S_{b+1} to be added to the end of the list has to be an f-neighbor of S_b with respect to n_u, i.e., $S_{b+1} \in N_f(S_b, n_u)$. As it is well known that flow, density and velocity are the three variables of traditional theory for uninterrupted traffic flow [84], we choose S_{b+1} among the base clusters in $N_f(S_b, n_u)$ by considering the *flow factor* q, *density factor* k and *speed limit factor* v.

Definition 9 Given a base cluster S and n_u as one endpoint of road segment e^S, the *flow factor* q, *density factor* k and *speed limit factor* v of a base cluster $S_j \in N_f(S, n_u)$

with respect to S are defined respectively as follows:

$$q = \frac{f(S, S_j)}{|PTr(S)|} \tag{1}$$

$$k = \frac{d(S_j)}{d(S) + \sum_{S_i \in N_f(S, n_u)} d(S_i)} \tag{2}$$

$$v = \frac{speed(S_j)}{\sum_{S_i \in N_f(S, n_u)} speed(S_i)} \tag{3}$$

where $speed(S_j)$ is the speed limit of e^{S_j}.

Definition 10 Given a base cluster S and n_u as one endpoint of e^S, the *merging selectivity* of a base cluster $S_j \in N_f(S, n_u)$ is defined as:

$$SF(S, S_j) = w_q \times q + w_k \times k + w_v \times v \tag{4}$$

where the coefficients w_q, w_k and w_v determine the weights of q, k, v respectively. The weights $w_q \geq 0$, $w_k \geq 0$ and $w_v \geq 0$ satisfy $w_q + w_k + w_v = 1$.

Here we assume that S is currently either the first element or the last element of a flow cluster F. Thus, selecting a base cluster to merge with F implies selecting a base cluster to merge with S. The three factors q, k, v are computed for each candidate base cluster S_j and normalized between [0,1] to better represent their relative capability of extending the flow cluster under examination to form a route with high continuity and density as our clustering objectives. Specifically, the flow factor q measures the relative netflow between S and S_j, thus, high q means that S_j maintains high continuity of the traffic flow when merging with S. The density factor k measures the relative density of S_j among neighboring base clusters at the road junction shared with S, namely n_u, which contributes to the density characteristic of the extended flow. The speed limit factor v measures how fast mobile objects can move within S_j compared to its neighbors at n_u, which is also a capable metric to find popular routes as mobile objects tend to travel following routes with either shortest distances

or shortest travel times. In Definition 10, we combine all three factors q, k, v under a weighing scheme to select the most relevant base cluster to merge with respect to our clustering objectives. According to the above definitions, the base cluster in $N_f(S, n_u)$ which has the highest merging selectivity, denoted by SF_{max}, will be chosen to merge with S.

Each base cluster which has been merged into a flow cluster is removed from B. We also use a threshold *minCard* for the trajectory cardinality of a flow cluster to filter out those flow clusters whose trajectory cardinalities are smaller than *minCard*. Finally, the condition to stop expanding the list $\{S_a S_{a+1} \dots S_{b-1} S_b\}$ at the end of the list is when S_b has no f-neighbor with respect to its endpoint n_u, i.e., $N_f(S_b, n_u) = \emptyset$. A similar condition is applied to stop expanding the list at the front. When both conditions are reached, we add the resulting flow cluster to $\mathcal{W}$, the output set of flow clusters in Phase 2. Then, we begin the next iteration of flow-based clustering with the remaining base clusters in the list B with the same process as described above, until all the base clusters are assigned to flow clusters, i.e., B becomes empty.

2.3.2.3 Decisions on f-neighbor Merging

Suppose we are at a merging step and the highest merging selectivity at this step is SF_{max}. We discuss the case when there are more than one base cluster which have the merging selectivity values of SF_{max} at a merging step. Suppose a base cluster S is at one end of an intermediate flow cluster F and S has m equally qualified neighbors $S_1, S_2, \dots, S_m$ to merge with respect to their merging selectivity values, i.e., $SF(S, S_1) = \dots SF(S, S_m) = SF_{max}$. In our *flow-based clustering algorithm* described above, we randomly pick one base cluster in $\{S_1, S_2, \dots, S_m\}$. We call it "random SF_{max}" merging scheme. Here we propose two schemes in which instead of randomly picking, we carefully consider the potential merging opportunity of each equally qualified base cluster, called "look-back" and "look-ahead" schemes respectively. Both

schemes focus on the flow property of the traffic stream.

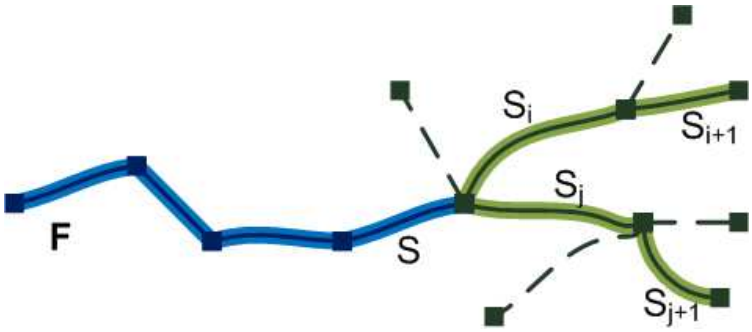


Figure 4: An example of a merging iteration in which a base cluster S at the end of an intermediate flow cluster F has two neighbors with $SF(S, S_i) = SF(S, S_j) = SF_{max}$

Look-back Merging. We compare the netflows between the flow cluster under examination and m base clusters $f(F, S_i), f(F, S_2), ..., f(F, S_m)$ and choose one $S_j (1 \leq j \leq m)$ such that $f(F, S_j) = max\{f(F, S_1), f(F, S_2), ..., f(F, S_m)\}$. This means we want to maintain the movement of as many objects which have already on the representative route of the flow cluster F as possible. So that we "look-back" to the flow cluster F which has been extended so far and pick among those m candidates the one that share the largest number of participating trajectories with F.

Look-ahead Merging. Let n_u be the intersection of e^S and e^{S_1}, e^{S_2},..., e^{S_m}. We consider the $maxFlows$ of those m candidates at the other endpoints, which are not n_u, of their representative road segments. If the $maxFlow$ of S_j has the biggest value among them, we will select S_j to merge with S. In this scheme, we "look-ahead" to the $maxFlow$ at the other endpoint of each candidate's representative road segment and select the largest one. By using "look-ahead" scheme, we want to make sure that we capture all the significant netflows while expanding a flow cluster.

An example of this case is illustrated in Figure 4 where a base cluster S at the end of an intermediate flow cluster F with $SF(S, S_i) = SF(S, S_j) = SF_{max}$. In the "random SF_{max}" scheme, S_i and S_j have equal merging opportunity, thus, we randomly pick either S_i or S_j to merge with S. In the "look-back" scheme, we pick

the base cluster which gives $max(f(F, S_i), f(F, S_i))$. In the "look-ahead" scheme, let S_{i+1} be a $maxFlow-neighbor$ of S_i and S_{j+1} be a $maxFlow-neighbor$ of S_j, we pick the base cluster either S_i or S_j based on which one gives $max(f(S_i, S_{i+1}), f(S_j, S_{j+1}))$.

2.3.2.4 Weight Assignment Criteria

The setting of the weights w_q, w_k and w_v is usually determined by the specific location-based applications. If an application favors the three factors equally when considering a traffic stream, we can set $w_q = w_k = w_v = 1/3$. If we set $(w_q, w_k, w_v) = (0,\ 1,\ 0)$, we will merge a base cluster with its densest f-neighbor. The resulting flows in this case will describe a route where traffic is highly concentrated. A combination of $(w_q, w_k, w_v) = (0,\ 0,\ 1)$ will produce flow clusters that describe the routes where objects can travel the fastest. For traffic monitoring applications, the flow factor and the density factor can be considered the most important factors so that we can set (w_q, w_k, w_v) to $(1/2,\ 1/2,\ 0)$. If the application emphasizes the flow property of a traffic stream and considers the flow factor as the most important one, it can set $w_q = 1$ and $w_k = w_v = 0$. Therefore, the $maxFlow$-neighbor of S (Definition 7) will be selected to merge with S.

Beside user supplied weights, NEAT also supports system generated weights to make it run in an automated manner. For system supplied weights, we introduce an *adaptive weight assignment* scheme which selects the most critical characteristics of the traffic stream based on the values of the flow, density and speed limit factors at each merging step. Suppose we are in the process of forming a flow cluster. At each merging step, we compute the maximum values q_{max}, k_{max}, v_{max} of the flow factor q, density factor k and speed limit factor v respectively. We the assign the weights w_q, w_k and w_v to be used at the corresponding merging step as follows:

$$w_q = q_{max}/(q_{max} + k_{max} + v_{max}) \tag{5}$$

$$w_k = k_{max}/(q_{max} + k_{max} + v_{max}) \tag{6}$$

$$w_v = v_{max}/(q_{max} + k_{max} + v_{max}) \quad (7)$$

Note that all three factors q, k, v are normalized between 0 and 1. Thus, their maximum values at each merging step q_{max}, k_{max}, v_{max} can show the relative importance of each property of the traffic stream passing the road intersection corresponding to the base cluster under consideration in the merging step. By adaptively assigning weights based on those maximum values as in Formula 5, 6, 7, NEAT puts more weight on the critical factor at each merging step and therefore, automatically discovers flow clusters which capture important traffic flow inherent from the given trajectory data.

The pseudo code of flow-based clustering is presented in Algorithm 1. It performs on a set of base clusters B until all the base clusters are assigned to flow clusters, i.e., B is empty (lines 3-11). Each round of flow-based clustering starts with the dense core of B (line 4). The procedure $expandFlow()$ (lines 19-32) performs f-neighbor merging to extend a flow from both sides (lines 8-9) given the factor weights w_q, w_k, w_v. If the input weights (w_q, w_k, w_v) are set to $(0, 0, 0)$, the adaptive weight assignment scheme is used to adaptively compute the weights at each merging step (line 21). It computes the f-neighborhood of the base cluster at one end of the flow (line 19) then select the f-neighbor with maximum merging selectivity to assign to the current flow. The flow continues to be expanded (line 31) as long as there are candidates to be merged (line 23). Otherwise, it stops. We also filter out the flows with insufficient number of participating trajectories. We set $minCard$ to equal to the average trajectory cardinality of the flow clusters in the set W. Those flow clusters whose trajectory cardinalities are smaller than $minCard$ will be discarded (lines 13-17).

2.3.3 Phase 3 - Flow Cluster Refinement

The third phase of NEAT is designed to exploit opportunities to further merge some flow clusters produced from Phase 2. We describe in this section a density-based

Algorithm 1 Flow-based clustering

Input: (1) A set of base clusters $B = \{S_0, S_1, ..., S_M\}$
(2) A road network G
(3) Factor weights w_q, w_k, w_v

Output: A set of flow clusters $W = \{F_1, F_2, ..., F_Q\}$

1: $flowId = 0$;
2: $W = \emptyset$;
3: **while** $B \neq \emptyset$ **do**
4: $S_c = densecore(B)$;
5: add S_c to F_{flowId};
6: remove S_c from B;
7: $\{n_{c1}, n_{c2}\}$ =getEndPoints(e^{S_c});
8: expandFlow($S_c, n_{c1}, flowId, w_q, w_k, w_v$);
9: expandFlow($S_c, n_{c2}, flowId, w_q, w_k, w_v$);
10: $flowId$++;
11: **end while**
12: $minCard$ =averagePtr(W);
13: **for each** $F_i \in W$ **do**
14: **if** Ptr(F_i) $< minCard$ **then**
15: remove F_i from W
16: **end if**
17: **end for**
18: Procedure **expandFlow**($S, n_u, flowId, w_q, w_k, w_v$){
19: N_f =getfNeigborhood(S, n_u);
20: **if** $(w_q, w_k, w_v) = (0, 0, 0)$ **then**
21: $(w_q, w_k, w_v) = getAdaptiveWeights(S, N_f)$;
22: **end if**
23: **if** $N_f \neq \emptyset$ **then**
24: **for each** $S_i \in$N$_f$ **do**
25: $SF(S_i)$ = mergeSelectivity(S_i, w_q, w_k, w_v);
26: **end for**
27: $SF(S_j) = max_{S_i \in N_f} SF(S_i)$;
28: add S_j to F_{flowId};
29: remove S_j from B;
30: n_v =getEndPoints(e^{S_j}) $\setminus \{n_u\}$;
31: expandFlow($S_j, n_v, flowId, w_q, w_k, w_v$);
32: **end if**
33: }

approach to group flow clusters. We modify the Hausdorff metric [16] to measure the distance between two flow clusters and adapt DBSCAN algorithm [31] to merge flow cluster merging process. This optimization is especially effective for real time

trajectory clustering where online clustering can be executed in an incremental and distributed manner. In particular, the first two phases of NEAT can be performed on each newly arrived set of trajectories. The new flow clusters are then merged with the available flow clusters to produce compact clustering results.

2.3.3.1 Distance Function for Flow Clusters

The Hausdorff distance is widely used in the area of computer vision (e.g., [48], [75]) to measure the resemblance of two sets of geometric points. Given two sets of points $\{A = a_1, a_2, \ldots, a_u\}$ and $B = \{b_1, b_2, \ldots, b_v\}$, the Hausdorff distance between A and B is defined as:

$$\begin{aligned} dist\_H\,(A, B) &= max\{max_{a \in A} min_{b \in B} ||a - b||, \\ & \quad\; max_{b \in B} min_{a \in A} ||b - a||\} \end{aligned} \tag{8}$$

where $||a - b||$ is the distance between two points a and b which is measured using some distance metrics (e.g., the Euclidean distance, the Manhattan distance [17]).

We modify the popular Hausdorff distance function to measure the distance between two flow clusters F_i and F_j in terms of network proximity. The distance between two flow clusters F_i and F_j can be determined by the distance between their representative routes r_{F_i} and r_{F_j}. In the first prototype of NEAT, we measure the distance between r_{F_i} and r_{F_j} by the network proximity of their ending locations. When the ends of two flow clusters are within a predefined network distance, we merge them into a larger cluster such that the resulting cluster will be able to show a group of frequent routes between two hotspot areas as illustrated in Figure 3.

Definition 11 Given two flow clusters $F_i \in \mathcal{W}$ and $F_j \in \mathcal{W}$, the distance between

F_i and F_j is defined as:

$$\begin{aligned} dist_N(F_i, F_j) &= dist_N\left(r_{F_i}, r_{F_j}\right) \\ &= max\{max_{a\in\{a_1,a_2\}} min_{b\in\{b_1,b_2\}} d_N(a,b), \\ &\quad max_{b\in\{b_1,b_2\}} min_{a\in\{a_1,a_2\}} d_N(b,a)\} \end{aligned} \tag{9}$$

where $d_N(a,b)$ is the shortest path from a to b, and $\{a_1, a_2\}$, $\{b_1, b_2\}$ are the two endpoints of r_{F_i}, r_{F_j} respectively.

2.3.3.2 Density-based Optimization

With the modified Hausdorff distance measure for flow clusters, we need an algorithm to merge the flow clusters when their distance is within some user-defined or system-supplied default threshold. We adapt the DBSCAN algorithm to group the set of flow clusters when the density opportunity exists. The DBSCAN algorithm was originally used to cluster a set of data points and requires two parameters: a distance threshold ε between two points and a minimum number of points *minPts* in a cluster. An object is a member of a cluster if it has at least *MinPts* neighboring objects within a given radius ε. All the objects in its ε-neighborhood are also members of the same cluster. Otherwise the object is classified as noise. We make the following modifications:

- The data unit to be clustered is a flow cluster. Thus the size of the dataset to be clustered in this phase is much smaller compare to the number of geometric points to be clustered in the base cluster formation phase or the number of base clusters of the same trajectory dataset to be clustered in the flow-based clustering phase.

- The distance function is our modified Hausdorff distance between two flow clusters.

- No minimum cardinality is set for the resulting cluster. A cluster with one flow

cluster is considered as good as others and should not be marked as noise and removed.

- The density based clustering for merging flow clusters always starts each round with the flow cluster whose representative route is the longest. This ensures that the final clustering results are always the same with the same distance threshold., which is different from the traditional DBSCAN in which data points are not processed in a deterministic order.

2.3.3.3 Flow-Distance Computation Optimization

As shown in Formula 9, a distance computation for each pair of flow clusters consists of four shortest path computations. Note that $d_N(a, b)$ and $d_N(b, a)$ are considered the same without loss of generality. Let $\mathcal{W} = \{F_1, F_2, \ldots, F_C\}$ $(C > 1)$ denote the set of flow clusters generated from Phase 2 of NEAT. For each F_i, we need to compare it with the rest $C - 1$ flow clusters, one at a time, to determine whether there is a merging opportunity. When the number of flow clusters in $\mathcal{W}$ is large, the cost of computing the network proximity of a pair of flow clusters can be expensive. If we use the popular Dijkstra's network expansion algorithm [29] to compute these shortest paths, the computation cost can be high compared to the standard Euclidean distance, especially when the representative routes of the flow clusters are long in terms of segment counts. For a graph with n nodes and m edges, traditional network expansion based algorithms (e.g., Dijkstra, Floyd-Warshall) compute shortest paths for each node pair in $O(nlogn + m)$ [81], while the Euclidean distance computation for a node pair only takes $O(1)$.

In phase 3 of NEAT, we use the Euclidean lower bound (ELB) [77] property to reduce the number of shortest path computations while retrieving the ε-neighborhood of a flow cluster F_i. By ELB, the Euclidean distance $d_E(l_i, l_j)$ between two road network locations l_i and l_j is always the lower bound of the network distance $d_N(l_i, l_j)$,

Algorithm 2 ε-neighborhood retrieval using ELB

Input: (1) A set of flow clusters $W = \{F_1, F_2, ..., F_Q\}$,
(2) A flow cluster $F_i \in W$,
(3) A road network G, (4) A distance threshold ε
Output: The ε-neighborhood N_ε of F_i

```
1:  N_ε = ∅;
2:  for each F_j ∈ W and F_j ≠ F_i do
3:     {a_1, a_2} = getEndPoints(F_i);
4:     {b_1, b_2} = getEndPoints(F_j);
5:     mind_E = +∞;
6:     for each a ∈ {a_1, a_2} do
7:        for each b ∈ {a_1, a_2} do
8:           if mind_E < d_E(a, b) then
9:              mind_E = d_E(a, b);
10:          end if
11:       end for
12:    end for
13:    if mind_E < ε then
14:       compute d_N(F_i, F_j);
15:       if d_N(F_i, F_j) < ε then
16:          add F_j to N_ε;
17:       end if
18:    end if
19: end for
```

i.e., the condition of $d_E(l_i, l_j) \leq d_N(l_i, l_j)$ is always hold. Hence, if $d_E(l_i, l_j) > \varepsilon$, we also have $d_N(l_i, l_j) > \varepsilon$. In case of $d_N(F_i, F_j)$, instead of computing four shortest paths $d_N(a_1, b_1)$, $d_N(a_1, b_2)$, $d_N(a_2, b_1)$ and $d_N(a_2, b_2)$, we compute four Euclidean between those locations first. If the minimum Euclidean distance between them exceeds ε, we can filter F_j from the search space for ε-neighborhood of F_i. Only when the minimum Euclidean distance between those points does not exceed ε, we calculate $dist_N(F_i, F_j)$ using our modified Hausdorff distance to determine whether F_j is in the ε-neighborhood of F_i or not. This is described in Algorithm 2. Algorithm 3 illustrates our DBSCAN adaptation to cluster a set of flow clusters W, given the road network G and a distance threshold ε. Function $getEpsNeighborhood(F_i, \varepsilon)$ in this algorithm implements Algorithm 2.

Algorithm 3 Density-based Optimizing

Input: (1) A set of flow clusters $W = \{F_1, F_2, ..., F_Q\}$,
(2) A road network G, (3) A distance threshold ε
Output: A set of trajectory clusters $O = \{C_1, C_2, ..., C_P\}$

```
1: clusterId = 1;
2: mark all F_i ∈ W as unvisited;
3: for each F_i ∈ W do
4:    if F_i is unvisited then
5:       assign F_i to cluster C_clusterId;
6:       mark F_i as visited;
7:       N_ε = getEpsNeighborhood(F_i, ε) ;
8:       expandCluster(N_ε, clusterId, ε);
9:       clusterId++;
10:   end if
11: end for
12: Procedure expandCluster(N_ε, clusterId, ε){
13: while N_ε ≠ ∅ do
14:   X = getFirst(N_ε);
15:   if X is unvisited then
16:      assign X to C_clusterId;
17:      mark X as visited;
18:      N'_ε = getEpsNeighborhood(X, ε) ;
19:      if N'_ε ≠ ∅ then
20:         for each X' ∈ N'_ε do
21:            if X' is unvisited then
22:               add X' to N_ε;
23:            end if
24:         end for
25:      end if
26:   end if
27:   remove X from N_ε;
28: end while
29: }
```

Table 1: Road networks used in our experiments

Regions	Total length	Segments	Junctions	Avg. seg. length	Junction degree
NW Atlanta, GA	1384.4km	9187	6979	150.7m	avg: 2.6, max: 6
W San Jose, CA	1821.2km	14600	10929	124.7m	avg: 2.7, max: 6
Mia-dade, FL	26148.3km	154681	103377	169.0m	avg: 3.0, max: 9

2.4 Experimental Evaluation

We perform five sets of experiments to evaluate the efficiency and effectiveness of our NEAT framework. Real road networks of different sizes are used in our experiments.

In order to analyze the performance of each phase, we refer to the trajectory clustering using Phase 1 of NEAT as the base-NEAT, the trajectory clustering using the first two phases as the flow-NEAT, and the trajectory clustering using all three phases as opt-NEAT. NEAT allows users to perform trajectory clustering using any of these three versions of NEAT. Base clusters, flow clusters and final trajectory clusters are the outputs of base-NEAT, flow-NEAT and opt-NEAT respectively, and each may have its own goal in terms of delivering interesting trajectory clustering results to location-based applications.

2.4.1 Experimental Setup

We use three real road networks in our experiments (Table 1). The road networks of North West Atlanta (ATL) and West San Jose (SJ) are obtained from [85]. The Miami-Dade (MIA) road network is obtained from [83]. We adapt the public event-based simulator GTMobiSIM [70] to generate thousands of mobility traces on those road networks for a large-scale evaluation. Each mobility trace is a sequence of road network location points. We use five trajectory datasets for each of the road networks ATL, SJ and MIA. Table 6 gives the information of our synthetic datasets. To create a trajectory dataset, for example SJ1000, we place 1000 mobile objects on West San Jose road network to travel under speed limit constrained on road segments, following shortest paths to a final destination chosen randomly from a predefined set of locations. We implement our algorithms using Java and visualize the results using GTMobiSIM GUI. All the experiments are conducted on the NEAT server machine with Intel Core2 Duo CPU of 2.00GHz and 1GB of main memory allocated for the Java heap size.

Table 2: Datasets used in our experiments

Datasets	**Number of points**		
	ATL	SJ	MIA
ATL/SJ/MIA500	114878	131982	276711
ATL/SJ/MIA1000	233793	255162	452224
ATL/SJ/MIA2000	468738	542598	893412
ATL/SJ/MIA3000	669924	794638	1302145
ATL/SJ/MIA5000	1277521	1296739	2262313

2.4.2 Visualization of NEAT clustering results

We visualize the clustering results obtained after processing trajectory datasets with the two phase NEAT approach (flow-NEAT) and with the three-phase NEAT approach (opt-NEAT). Figure 6 and Figure 7 show the clustering results for ATL500 dataset. Figure 5 plots 500 trajectories (in green color) on North West Atlanta map. After the first two phases, 31 flow clusters are discovered (Figure 6). These flow clusters capture all the major traffic flows from the ATL500 dataset. Some traces that we see in the original dataset disappear in Figure 6 since there are too few objects moving in them. The threshold to filter those flow clusters in this experiment is $minCard$=5, which is the average number of participating trajectories in each of the flow clusters. There are two dense regions that concentrates the short flows. They are the two hotspots where we place the 500 mobile objects at the beginning of their trips. After travelling on those short flows, they start merging into the long flows to reach one of the three destinations marked with the red X-signs on the map. These 31 flow clusters are grouped into 2 clusters as shown in Figure 7 after performing the density-based flow cluster refinement. We start the density-based optimization with the longest representative route (the dark red polyline number 30 in Figure 6). This route connects to one of the two hotspots and its endpoints are close to the endpoints of the other long flows (the polylines 29, 28 and 27). As defined in the DBSCAN algorithm, they are in a *density-connected* set. Therefore, when we perform density-based clustering (with the distance threshold ε = 6500m) we have them grouped

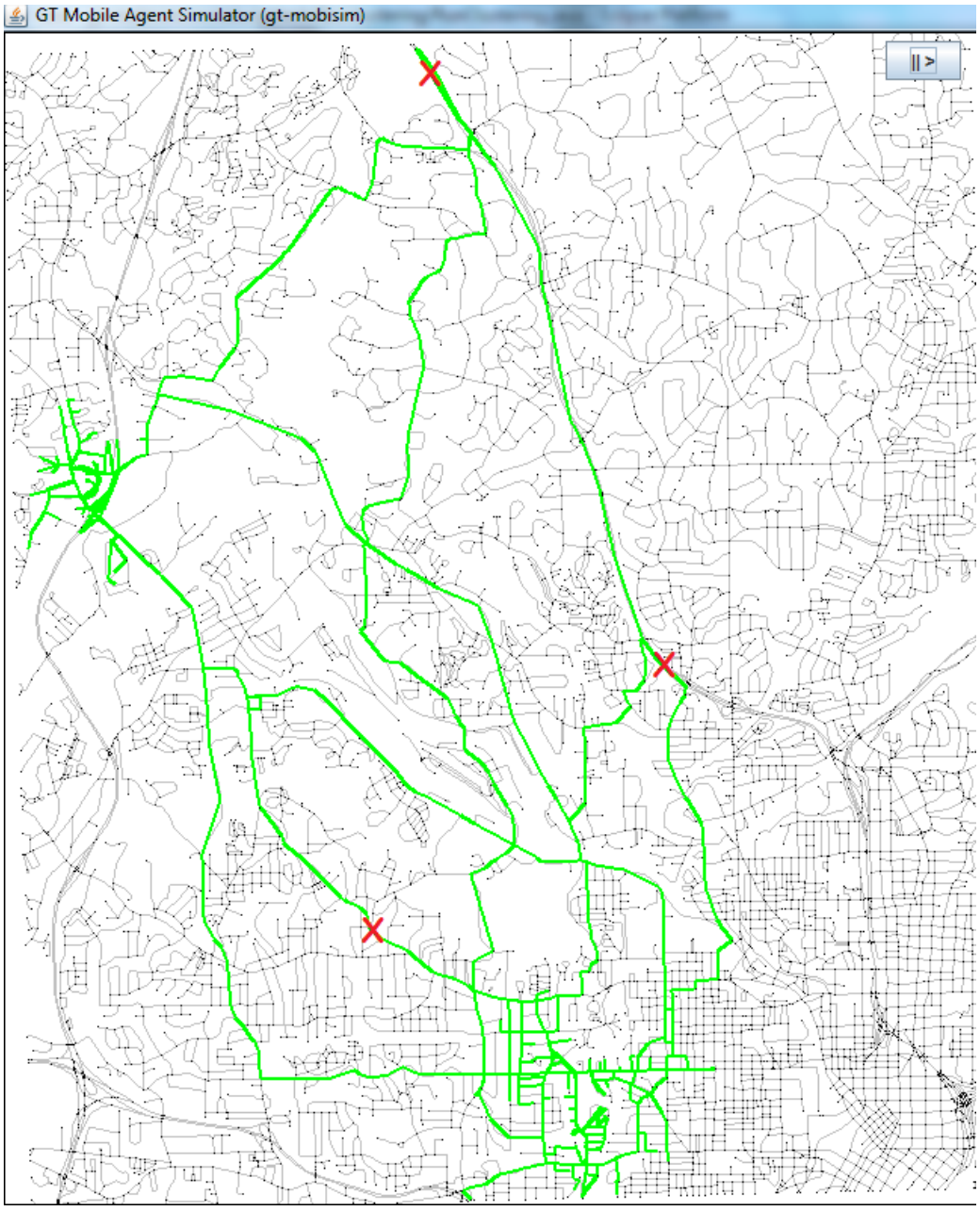


Figure 5: Input data: ATL500

together in one cluster (contains the red polylines in Figure 7). The rest of the flows are grouped into another cluster (contains the gray polylines). The results for West San Jose road network and Miami-Dade datatsets are shown in Figure 8 and Figure 9. For SJ5000, flow-NEAT produces 172 flows (Figure 8(a)) and opt-NEAT produces 13 clusters with $\varepsilon = 1200$m (Figure 8(b)). For MIA3000, flow-NEAT produces 300

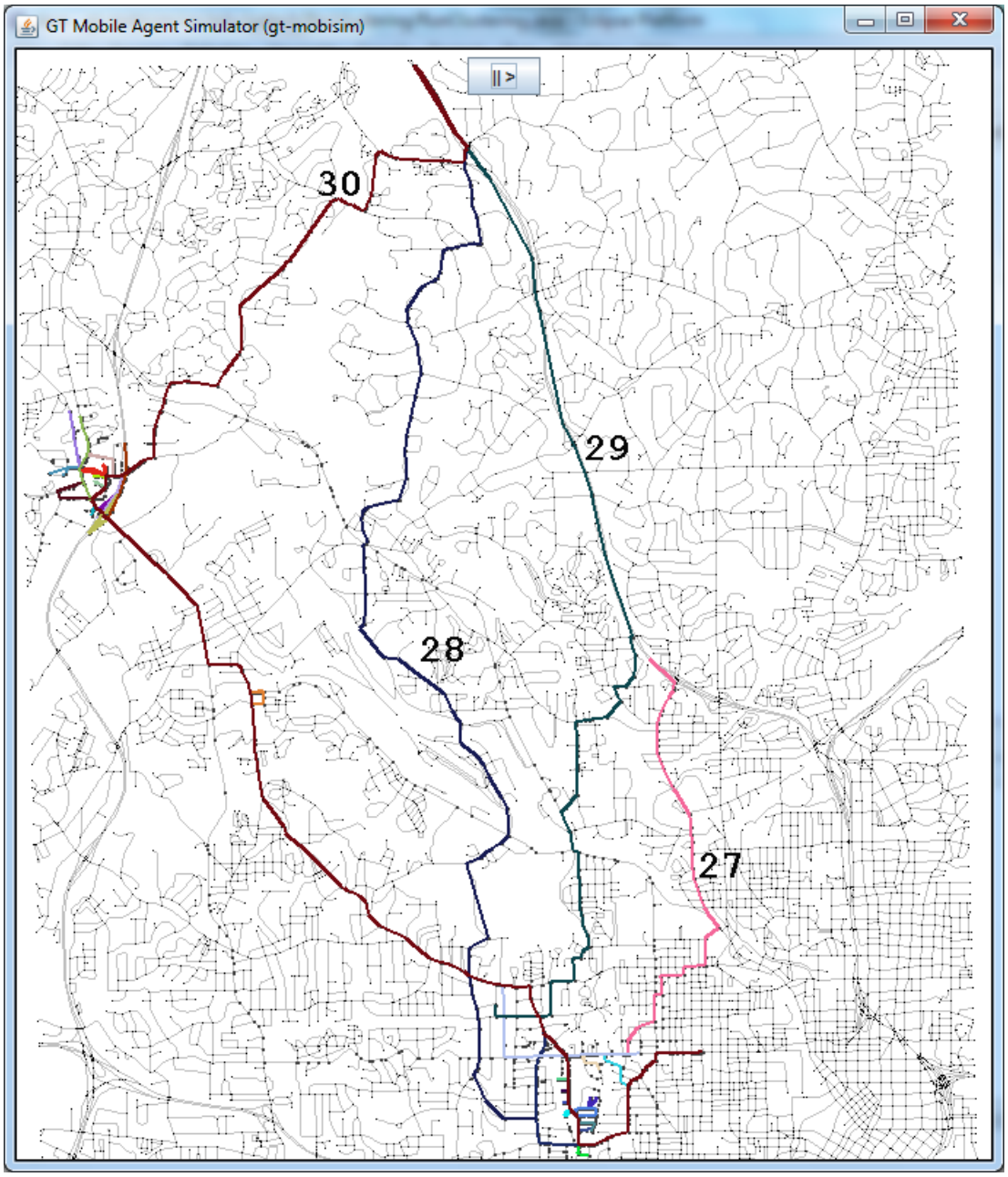


Figure 6: Result for ATL500: 31 flow clusters

flows and opt-NEAT produces 33 clusters with $\varepsilon = 2000$m. The visualization for MIA3000 (Figure 9) is not as clear as the other two datasets because Miami-Dade road network, which is an urban area, is more dense and complicated than the ATL and SJ road networks (see Table 1 and 6).

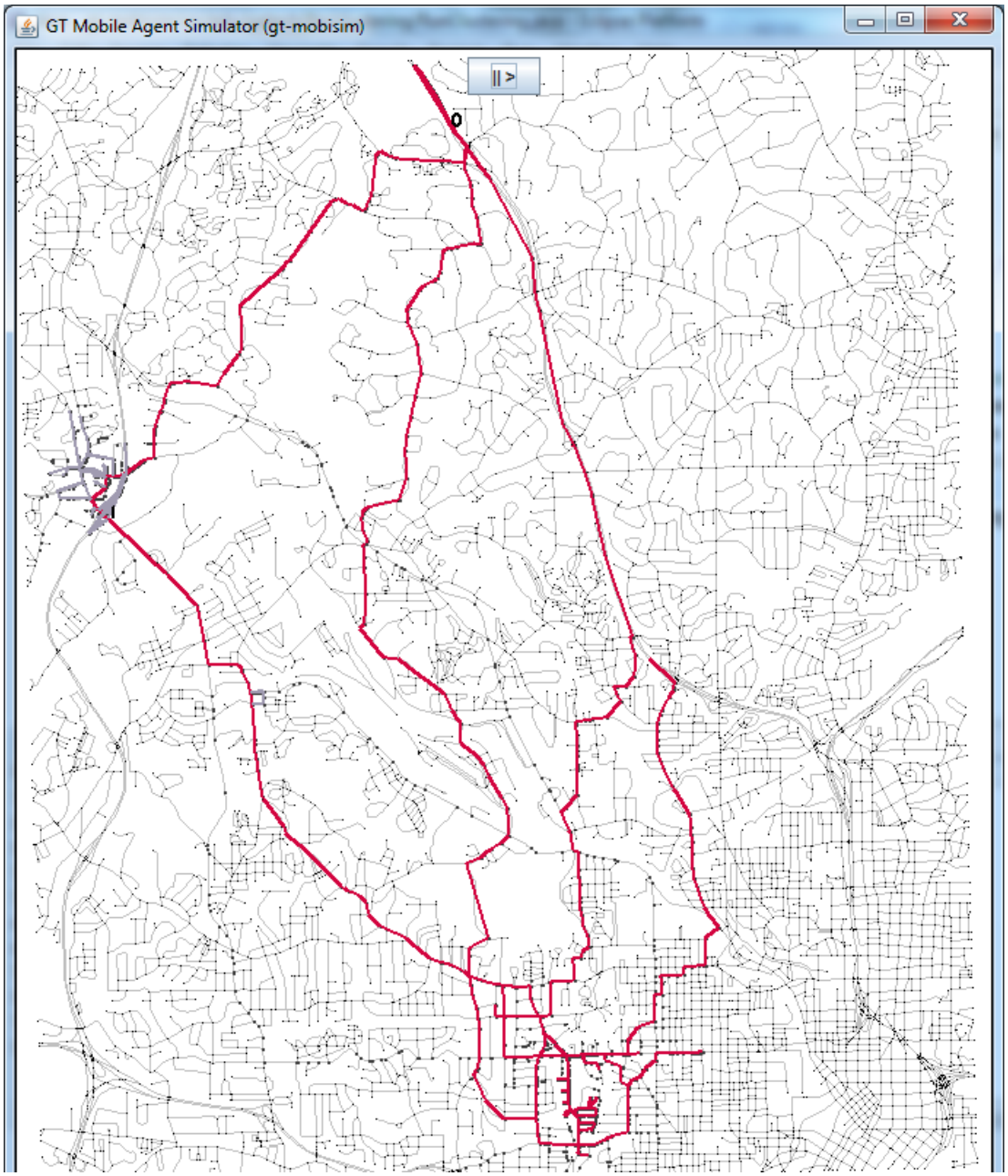
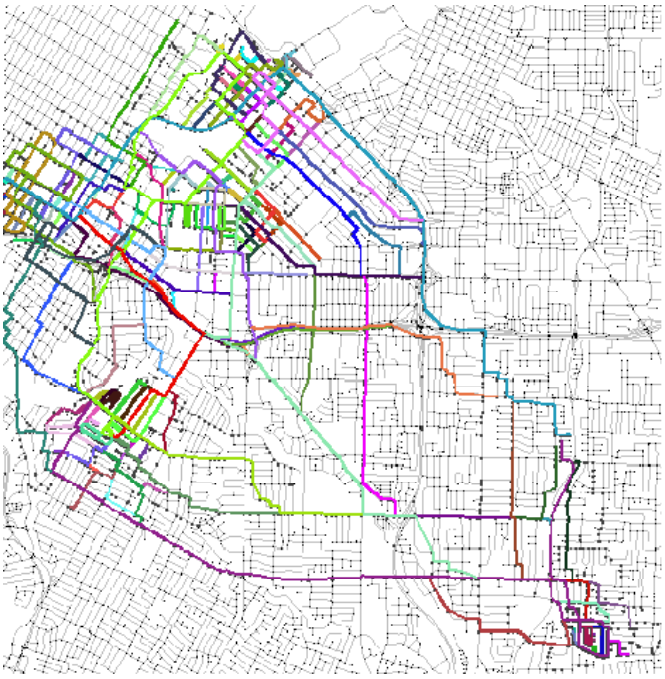


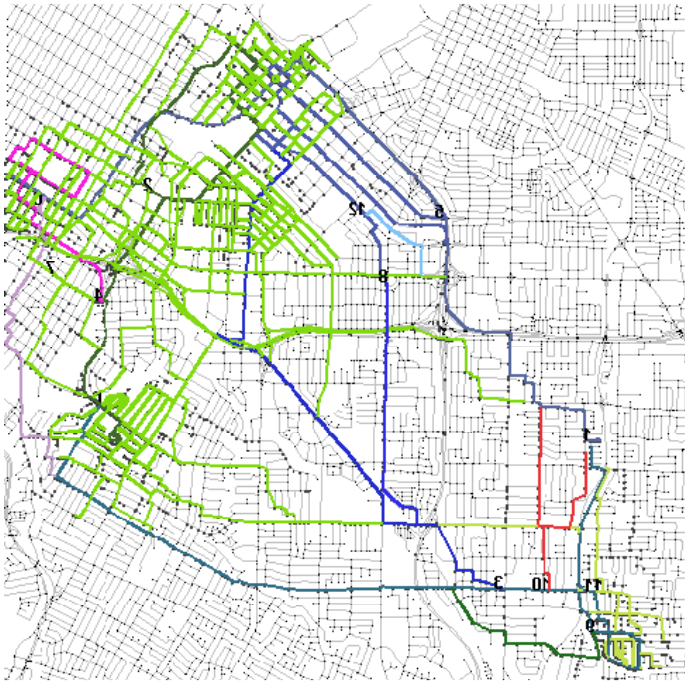
Figure 7: Result for ATL500: 2 clusters after optimizing phase ($\varepsilon = 6500$)

2.4.3 Efficiency and Effectiveness of NEAT

In this section we evaluate the efficiency and effectiveness of NEAT by comparing it with the conventional density-based approach, represented by TraClus. Figure 10 shows 81 resulting clusters when applying the TraClus algorithm with $\varepsilon = 10$m and $MinLns = 30$ to cluster ATL500 dataset. Each cluster has its representative



(a) 172 flow clusters for SJ5000



(b) 13 clusters after optimizing phase (SJ5000, $\varepsilon = 1200$)

Figure 8: Results for West San Jose road network

trajectory plotted and numbered on the map. Another result of 460 clusters for ATL500 is shown in Figure 11 when applying Traclus with ε = 1m and $MinLns$ =

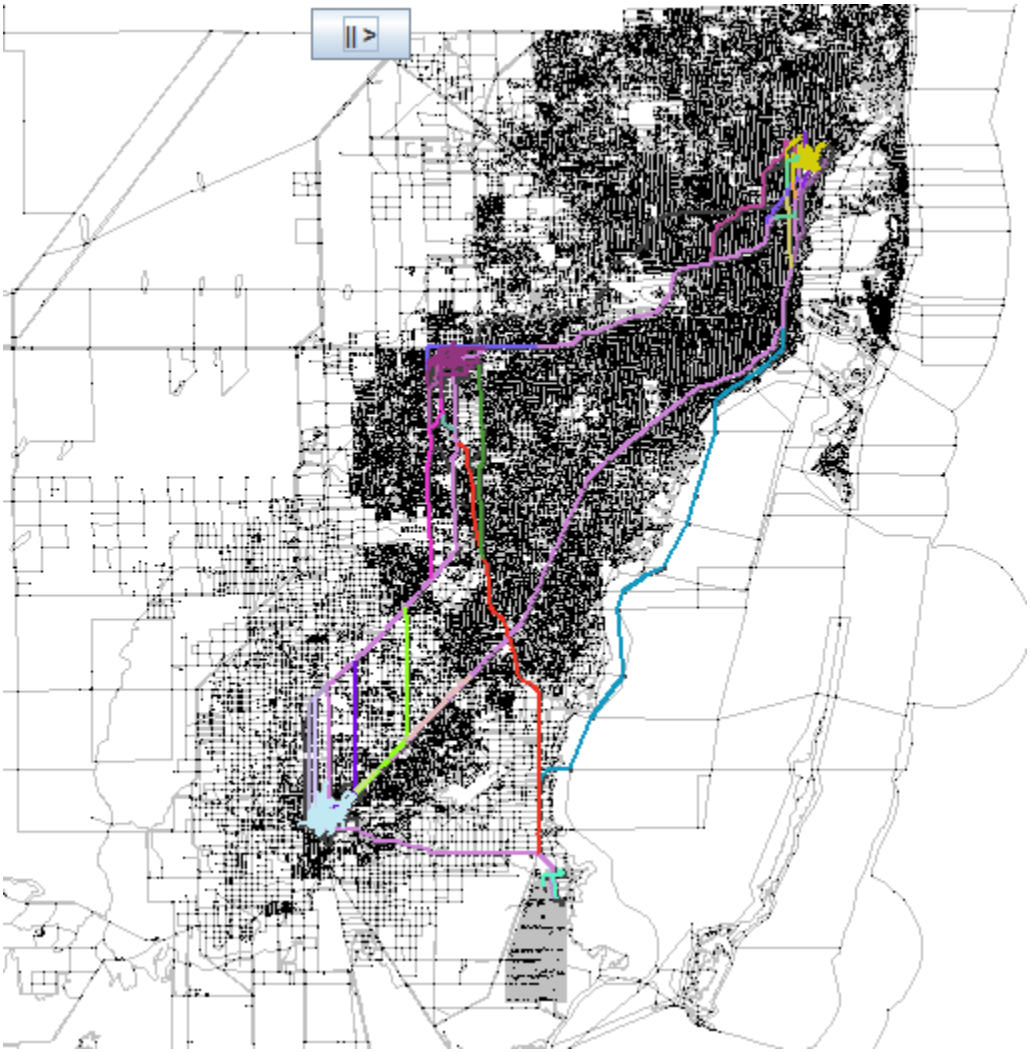


Figure 9: Result for MIA3000: 2 clusters after optimizing phase ($\varepsilon = 6500$)

1. As we see, these clusters are discrete on the road network. Their representative trajectories are short in lengths. These clusters only show short routes in the road network where there is dense traffic. They do not provide information about the traffic continuity implied in the original trajectory dataset. Recall the NEAT clustering results shown in Figure 6 and Figure 7, we can see that most of the important routes are missed when using TraClus. An interesting point to note is that our framework with base-NEAT can also provide this knowledge if we filter out those base clusters where the density is below a specific threshold. The remaining base clusters will represent the road segments where traffic is highly concentrated.

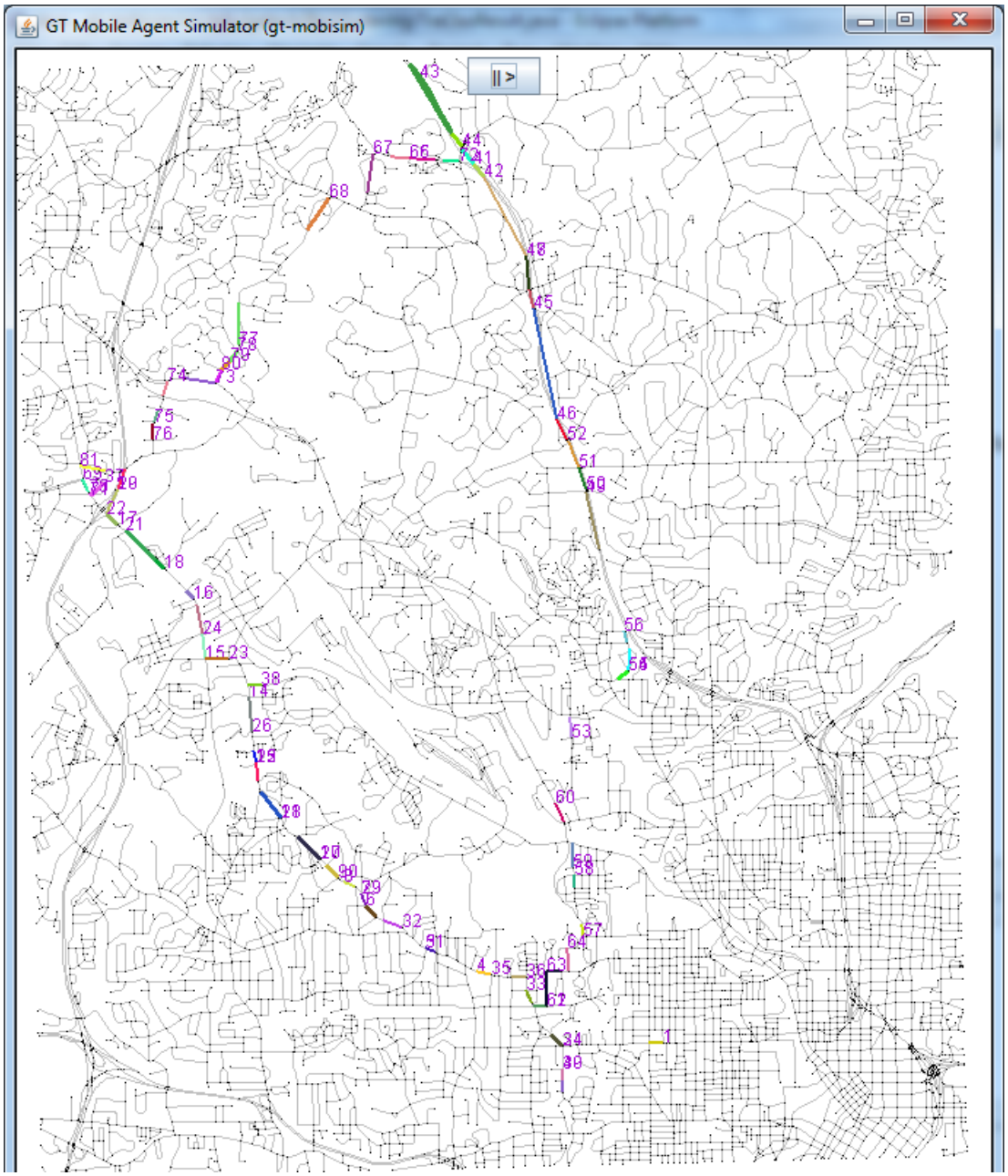


Figure 10: TraClus: 81 clusters for ATL500 (ε=10m, $MinLns$=30)

Figure 12(a) and Figure 12(b) shows the comparisons of the average and maximum lengths of the representative routes discovered using flow-NEAT and TraClus. Compared to TraClus, flow-NEAT produces clusters with longer representative routes, which are favorable for location based applications such as bus line organizing or ride sharing [51]. This results in a smaller number of clusters produced by flow-NEAT

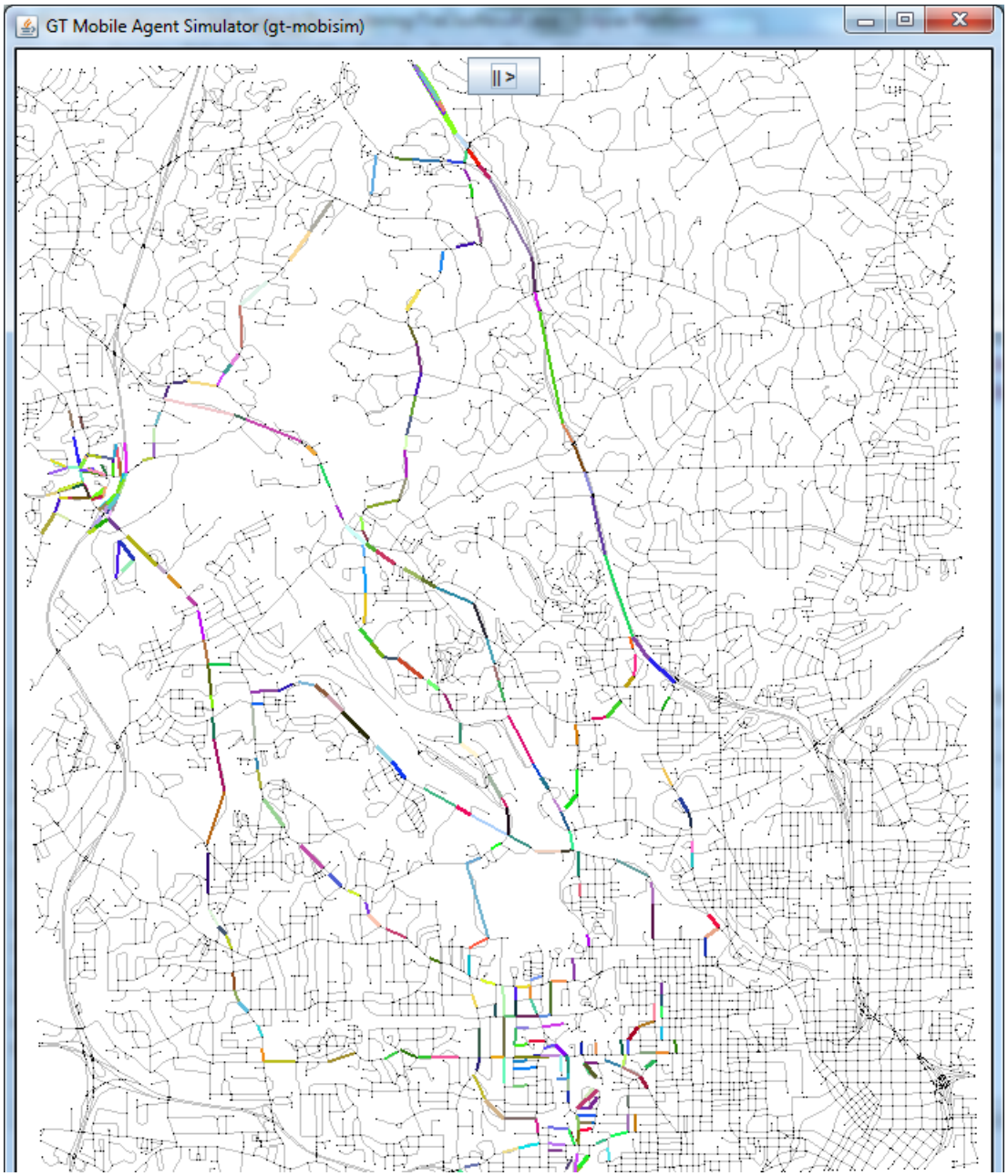
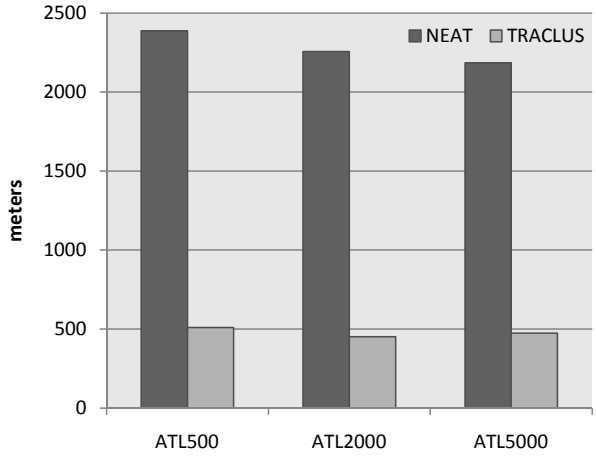


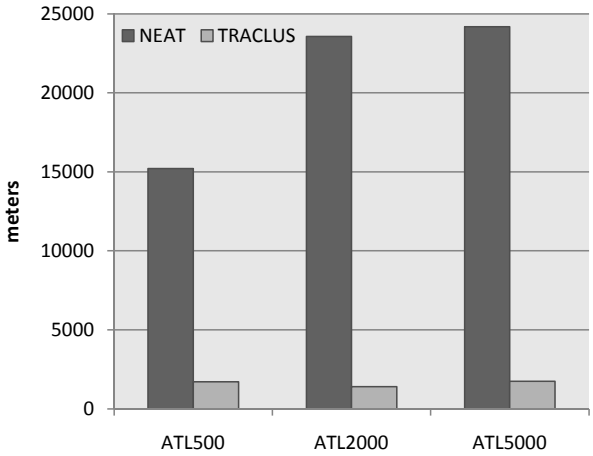
Figure 11: TraClus: 460 clusters for ATL500 (ε=1m, $MinLns$=1)

as shown in Fig 13. In comparison, NEAT produces more compact and meaningful results through road network aware trajectory clustering.

By utilizing the road network information, NEAT not only produces meaningful trajectory clusters but also runs very fast. Both traffic flow and traffic density can be discovered using flow-NEAT without the need to compute any distance function. We



(a) Average length



(b) Maximum length

Figure 12: Comparison of lengths of the representative routes discovered by flow-NEAT and TraClus (ATL datasets)

only compute shortest path distances in Phase 3 for clustering refinement and optimize this costly operation by using ELB filter to eliminate the unnecessary shortest path computation. In constrast, TraClus depends heavily on the distance measurements among every pairs of samples in the trajectory dataset. This makes TraClus overall very slow as the number of samples in each trajectory and the number of trajectories in each dataset are high. Figure 14 shows the efficiency comparison between the three-phase NEAT framework and TraClus framework by varying the number of points in ATL datasets. TraClus is very time-consuming and the time complexity

grows as the number of points gets larger. TraClus runs in 2573.5 seconds to cluster ATL500 (114878 points) and 334735.1 seconds (nearly 4 days) to cluster ATL5000 (1277521 points). While opt-NEAT only takes 1.29 seconds to cluster ATL500 and 59.7 seconds to cluster ATL5000. Compared to TraClus, the NEAT framework is faster by more than three orders of magnitude.

One may ask *what if TraClus is given the benefit of our map-matching preprocessing step to partition a trajectory into trajectory fragments and uses a network distance measure such as our modified Hausdorff function in its grouping phase?* To address this concern, we have run a variant of TraClus on our test datasets. In this variant of TraClus, we even provide TraClus with the partitioning of trajectories into base clusters instead of t-fragments, then the grouping phase merges the base clusters using our modified Hausdorff distance. Note that the number of base clusters is usually much smaller than that of t-fragments. However, TraClus remains slow compared to NEAT due to their grouping algorithm which heavily depends on distance computations and the resulting clusters only show discrete traffic density in the road network. For instance, with the SJ2000 dataset (226151 t-fragments, 901 base clusters), this variant of TraClus took 6396.79 seconds to finish with 117 resulting clusters. While NEAT produced a more compact results of 42 flow clusters and 14 final clusters in only 11.68 seconds.

2.4.4 Performance of NEAT Algorithms

We analyze the efficiency of NEAT by analyzing the performance of different versions of NEAT during the three phases, focusing on the impact of the flow property of traffic streams in NEAT design. The scaling of base-NEAT, flow-NEAT and opt-NEAT for different MIA datasets are shown in Figure 15(a). The curves are almost linear with the growth of dataset size. In all cases, the flow cluster refinement phase contributes very little to the total running time, as shown in the graphs, due to the ELB based

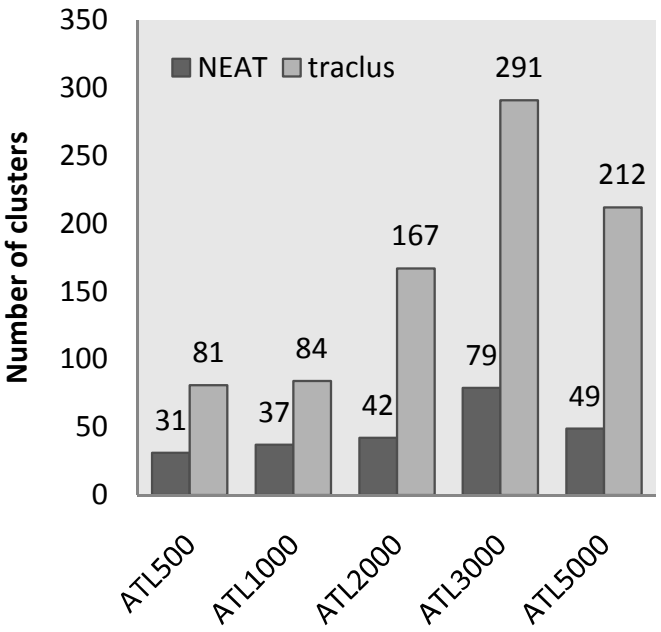


Figure 13: Numbers of resulting clusters using flow-NEAT and TraClus

Table 3: Number of flow clusters produced by opt-NEAT

Datasets	**SJ500**	**SJ1000**	**SJ2000**	**SJ3000**	**SJ5000**
# flows	73	156	55	52	180

optimization. The opt-NEAT curves nearly overlap the flow-NEAT curves.

We further investigate the relative performance of Phase 1 (base cluster formation) and Phase 2 (flow cluster formation). Phase 1 algorithm takes the road network locations as its data units because it scans the sequence of points in all the trajectories to extract t-fragments. Phase 2 algorithm takes the base cluster as its data unit. Intuitively, the number of road network locations in the trajectory dataset is much larger than the number of base clusters produced from Phase 1. Thus, it takes longer to complete Phase 1. This is confirmed by our experiments shown in Fig 15(b). Figure 16 unveils the effectiveness of using ELB to reduce the number of shortest path computations (opt-NEAT-ELB) versus using Dijkstra's network expansion algorithm to compute all the shortest paths (opt-NEAT-Dijskstra) when performing density-based optimization in the NEAT framework. In Figure 16(a), the opt-NEAT-Dijskstra

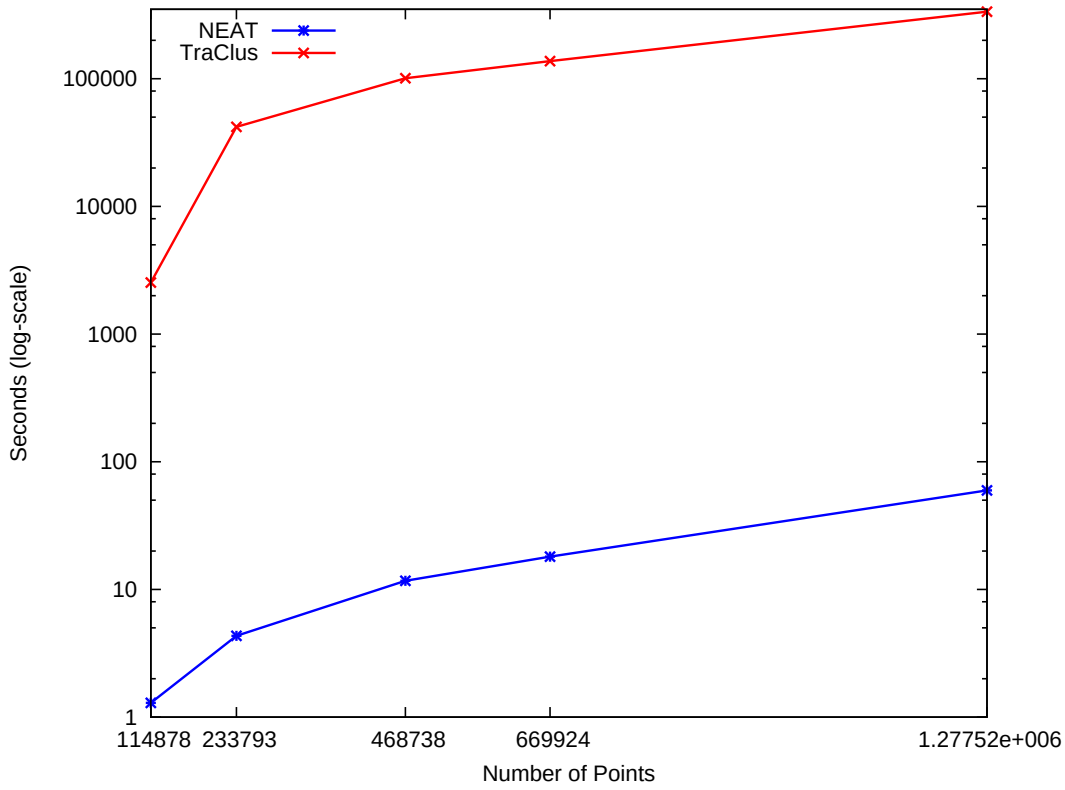
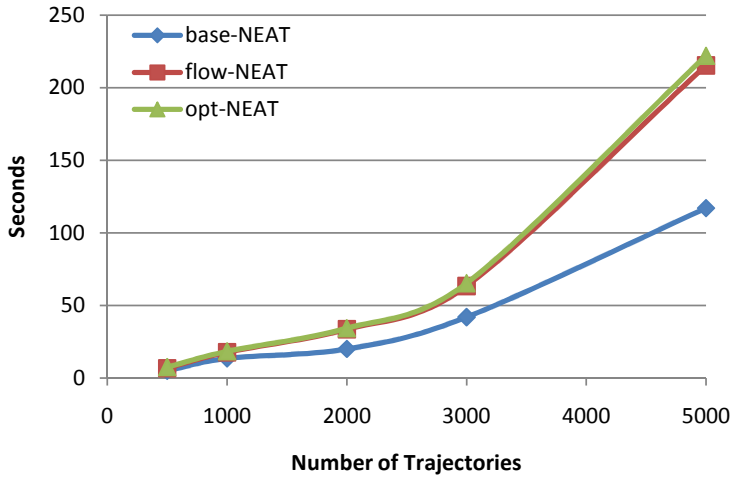


Figure 14: Running time comparison

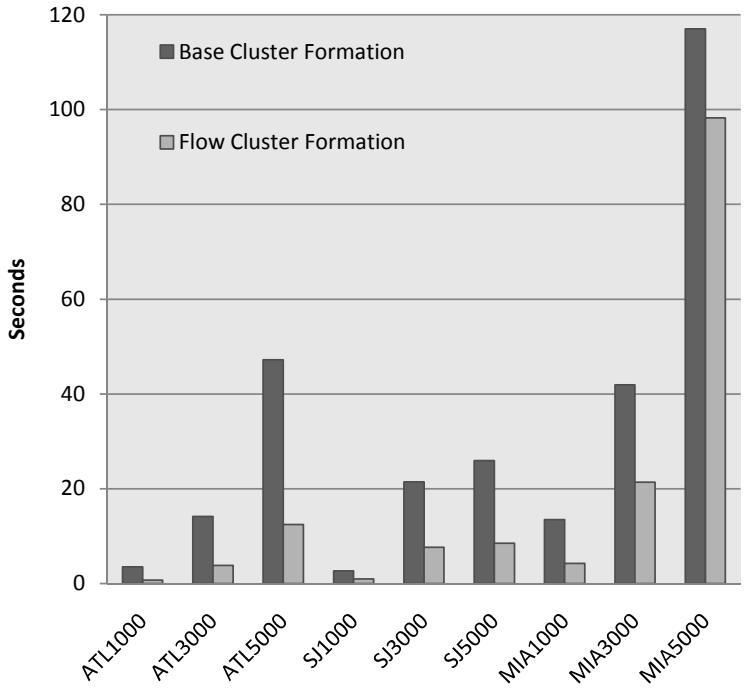
curve grows faster as the dataset size grows. However, the curve of opt-NEAT-Dijkstra in Figure 16(b) shows that the cost at SJ1000 are much higher than at SJ2000 and SJ3000. This is due to the cost of Phase 3, which computes shortest path distances, actually depends on the number of flows produced by Phase 2 and not the data size. Table 3 shows the number of resulting flows output from the second phase where numbers of flows in SJ1000 and SJ5000 are much higher than that in other datasets for SJ road network. Using ELB significantly speeds up the performance of the density-based optimization algorithm (see some big gaps between the two curves opt-NEAT-ELB and opt-NEAT-Dijkstra).

2.4.5 Effects of f-neighbor Merging Decisions on Flow-based Clustering

We study how using different merging schemes including "random SF_{max}" ("random" for short), "look-back" and "look-ahead" schemes, as described in Section 2.3.2.3,



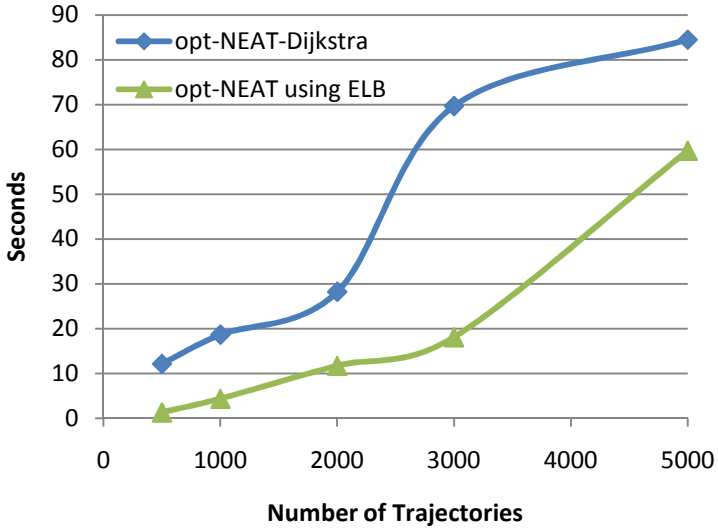
(a) MIA datasets



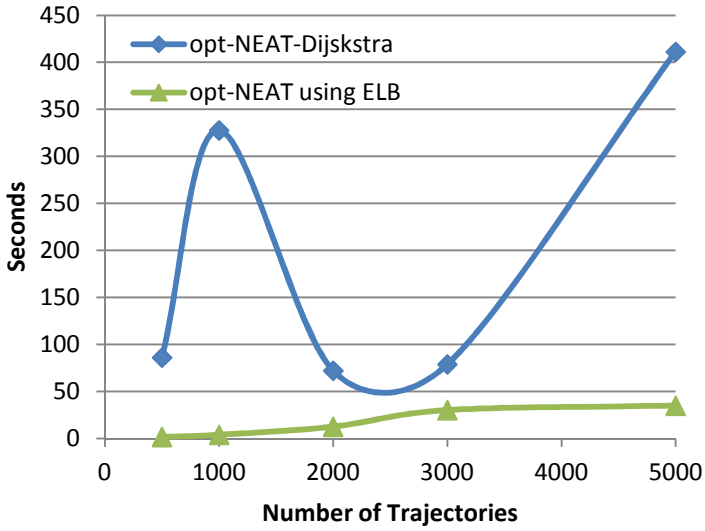
(b) Relative performance of base cluster formation and flow cluster formation

Figure 15: Relative performance of our approaches

affects the results of flow-NEAT. We measure the cluster quality by computing the total netflows of the resulting flow clusters. The results are reported for ATL and SJ datasets in Figure 17 including the running time (Figures 17(a) and 17(b)) and total netflows (Figures 17(c) and 17(d)) of each scheme. It is shown that in most cases, look-head merging runs faster than look-back scheme and random merging is



(a) ATL datasets



(b) SJ datasets

Figure 16: Effectiveness of using Euclidean lower bound

the fastest. Although the cost incurred by running look-ahead or look-back schemes compared to random merging is not significant, the total netflows of the resulting flow clusters in all three schemes are approximately the same. This results from the low junction degree of road networks where the average junction degree is usually less than 3. We conclude that in the context of a road network, "random SF_{max}" merging is good enough to get good clustering result and thus, NEAT uses "random SF_{max}"

merging in Phase 2.

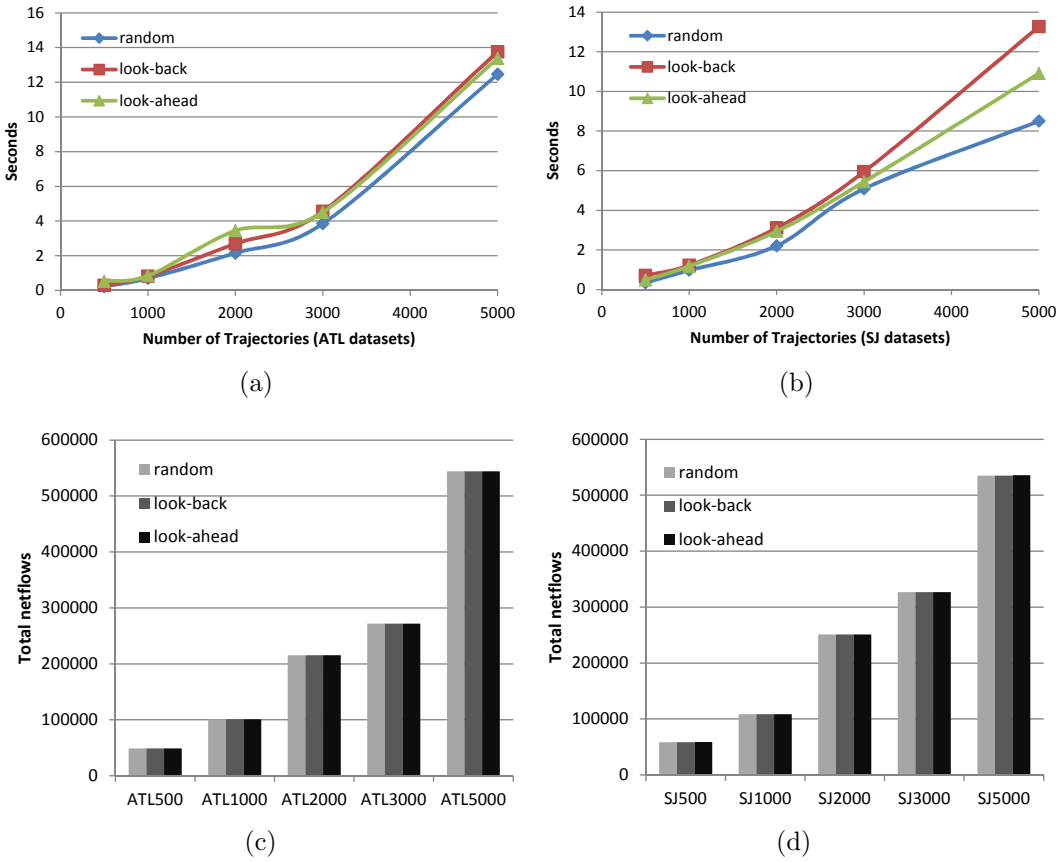


Figure 17: Effects of using random, look-back and look-ahead f-neighbor merging schemes

2.4.6 Effectiveness of Adaptive Weight Assignment on Flow-based Clustering

In this section, we evaluate the effectiveness of our *adaptive weight assignment* scheme (Section 2.3.2.4) compared to some predefined combinations of the weights (w_q, w_k, w_v) for flow factor, density factor and speed limit factor respectively. The different assignments of (w_q, w_k, w_v) represent different merging criteria when we perform f-neighbor merging. In the previous experiments, we want to focus on the flow property of a traffic stream so we set the highest weight for the flow factor ($w_q = 1$ and $w_k = w_v = 0$), i.e., a base cluster will always be merged with its $maxFlow$-neighbor. Since the speed limit is fixed for each type of road, the speed limit factor is the least favorable among

Table 4: Total netflows

	(1/2,1/2,0)	adaptive	(1,0,0)
ATL3000	225713	271782	271986
SJ3000	275665	326723	326870
MIA3000	812505	921476	921910

Table 5: Total cluster density

	(1/2,1/2,0)	adaptive	(0,1,0)
ATL3000	225713	374234	374268
SJ3000	451783	475508	475536
MIA3000	1241101	1276109	1277511

three factors. In our experiment with the adaptive weight assignment, we skip the speed limit factor by setting v_{max} value at each merging step to 0. We run flow-NEAT with this "adaptive weights" scheme and three predefined weight settings of $(1, 0, 0)$, $(0, 1, 0)$ and $(1/2, 1/2, 0)$. The $(1/2, 1/2, 0)$ weight assignment is used as a base setting where both flow and density are weighed equally. We measure the total netflows of the discovered flow clusters by running flow-NEAT using the "adaptive weights" scheme, compared to using the "netflows only" $(1, 0, 0)$ weight setting. The results are reported in Table 4. For ATL3000 dataset, our adaptive weights scheme achieves an increase of total netflows of 18.52% compared to the base settings $(1/2, 1/2, 0)$, while that of the $(1, 0, 0)$ setting is 20.41%. Similarly for SJ3000 and MIA3000, the percentage of total netflows increased using adaptive weights scheme is also so close to that of $(1, 0, 0)$ setting. With the same three datasets, Table 5 reports the total cluster density of the discovered flow clusters by running flow-NEAT using the "adaptive weights" scheme, compared to using the "density only" $(0, 1, 0)$ weight setting and $(1/2, 1/2, 0)$ setting. In all cases, the percentage of total cluster density increased using our adaptive weights scheme compared to the base setting of $(1/2, 1/2, 0)$ is very close to that of the $(1, 0, 0)$ setting (the difference is below 0.1%). Therefore, our adaptive weight assignment scheme can automatically captures highly dense and continuous traffic flow from trajectories in the road network.

2.5 Related Work

The clustering problem has been extensively researched in mobile ad hoc networks (MONET) and data streaming systems. In MONET, data unit to be clustered is the mobility node where the characteristics of a node are taken into account for cluster head choices in order to conserve energy and connectivity [23] [88] [90]. In data streams, k-means and density-based clustering algorithms have been extended to cluster large volume of multi-dimensional data points generated by sensor networks [8] [26] [73].

Most existing work on *MO* trajectory clustering [36] [66] [91] [56] [74] [59] [58] [11] has derived proximity measures for trajectories and adapted traditional k-means, hierarchical, or density-based clustering algorithms to group similar trajectories. Trajectory-OPTICS [66], which extends OPTICS algorithm [13], is a good example for grouping similar trajectories as a whole. The distance between two trajectories is the average Euclidean distance between two objects for every timestamp. Traclus [36] aims to find similar sub-trajectories rather than the whole trajectories. It partitions each trajectory into line segments using the Minimum Distance Length (MDL) principle and then performs a DBSCAN-like [31] clustering on line segments. The similarity measure is composed of three Euclidean based distance components between line segments. As a result, discovered clusters are dense regions of line segments. [59] adapts TraClus for online trajectory clustering. [58] extends TraClus for trajectory classification. However, these density-based methods cluster free space trajectories, i.e., without considering the constrained road network. Other works [91] [56] [74] consider the network constraint to derive similarity measures but they only focus on the density aspect of the given trajectories such as object density [56], common road segments [91], shortest path distance [74]. Our approach can produce partial clustering but carefully considers the constrained road network focusing on both flow and density characteristics and avoids the expensive shortest path computation in its first

two phases. NEAT innovates TraClus framework in a creative manner with higher efficiency (due to reduction of network distance computation) and higher accuracy (due to incorporation of flow semantics). This chapter provides a full-fledged development of the initial NEAT approach [40] with a thorough study on different merging decisions and an adaptive parameter assignment scheme capturing the most critical characteristics of a traffic stream in the core of NEAT, which allows it to discover important flow clusters in an automated manner.

2.6 Conclusion

We have presented NEAT - a novel road-network aware approach to *MO* trajectory clustering which clusters *MO* trajectories in a comprehensive three phase framework. We introduce the concept of f-neighborhood to identify the most critical and most interesting parts of the given *MO* trajectories with respect to clustering. Instead of taking points or line segments as the clustering unit as in traditional approaches, we introduce t-fragment, base cluster and flow cluster as the basic building blocks for road-network aware trajectory clustering. NEAT carefully combines the traffic locality, flow and density metrics in its three-phase trajectory clustering framework that significantly reduces the data space in each subsequent phase and ensures trajectory clustering quality. By carrying out extensive experiments, we show that NEAT discovers clusters of *MO* trajectories which represent major traffic stream in a road network and outperforms conventional density-based trajectory clustering algorithms in terms of both time complexity and accuracy.

CHAPTER III

TRACEMOB: A METHODICAL APPROACH TO CLUSTERING WHOLE TRAJECTORIES OF MOBILE OBJECTS IN ROAD NETWORKS

Most of mobile object trajectory clustering analysis to date has been focused on clustering the locations or subtrajectories extracted from trajectory data. This chapter presents TraceMob, a methodical approach to clustering analysis of whole trajectories of mobile objects traveling in road networks. TraceMob as a whole trajectory clustering framework has three unique features. First, it designs a quality metric to measure the distance between two whole trajectories. By quality we mean that the distance measure between two whole trajectories should minimize the bias of long or short trajectories, and should also capture the movement paths that are in close vicinity such as traveling paths of close-by parallel road segments. Second, we develop an algorithm that transforms whole trajectories in a road network space into multidimensional data points in an Euclidean space while preserving their relative distances in the transformed metric space. We achieve this objective by leveraging FastMap to carefully select the initial reference points and iteratively define the dimensionality d for the transformed metric space. Third, we develop a cluster validation method for evaluating the clustering quality in both projected trajectory image space and the road network space. Extensive experimental evaluation with trajectories generated on real road network maps of different cities shows that TraceMob produces higher quality clustering results and runs several orders of magnitude faster than the existing approaches.

3.1 Introduction

With advances in positioning technologies and the proliferation of Wifi/GPS-enabled smartphones, tablets and other handheld devices, we have witnessed an escalation of web-based and mobile location-aware applications with a torrent of location data, such as Google Maps, Bing Maps, Google Latitude, Apple's FindMyFriends, FourSquare, to name a few. We can classify mobile object trajectory-based research, applications and services into three categories based on what information about trajectories is utilized in trajectory analysis. The first category analyzes trajectory data as position points rather than time series of locations and offers algorithms to query and mine point-based location data [21, 27, 65, 87, 95, 96, 99], for example, to find nearby points of interests or discover hot-spot locations where people like to gather during weekends and holidays. The second category focuses on identifying interesting sub-trajectories from the datasets of whole trajectories based on density or flow patterns of mobile objects [36, 40]. For example, with subtrajectory clustering, one can discover the congestion patterns on the segment of W Peachtree Street NE between North Avenue and 14th Street in Atlanta city. The third category of trajectory clustering analyzes and mines the whole trajectories of mobile objects. It treats trajectories, the time series of location points recorded along traveling paths of mobile objects on a road network, as units of data analysis. The purpose of clustering whole trajectories is to discover the grouping structure in a given trajectory dataset. Each trajectory cluster represents a spatial trend in motion or movement behavior of mobile objects, revealing valuable information about potential social connections and common interests of mobile users moving in a road network. For example, a trajectory cluster including a user's past trajectories can be used as the prospective search space for her location based service requests. Whole trajectory clustering can provide better reference data for transportation planning based on the trajectory patterns and the traffic density in each of the trajectory clusters for more cost-effective road design.

Most of existing research on trajectory analysis belong to the first category − clustering location data points extracted from mobile object trajectories, and some of recent projects fall into the second category, represented by discovering subtrajectories of high density based on Euclidean distance [36] or clustering subtrajectories with significant traffic flows in addition to road-network distance based density [40]. However, few research efforts have engaged in clustering whole trajectories to date for a number of reasons. First, whole trajectories are time series of locations with different lengths (i.e., the number of locations per trajectory) and thus, any distance measure that relies on equal size trajectories may bias long trajectories over short ones even with regression methods that stretch short trajectories to the same size as the long ones. Second, trajectories are a special type of time series data that are constrained by the road network since mobile objects can only move along road segments and turn at road intersections. Thus, they may overlap with one another partially with respect to the road segments. However, some partially overlapped trajectories represent distinctly different trajectory clusters when grouping full trajectories. Also, non-overlapping trajectories may be close to one another semantically or based on road network distance (e.g., mobile objects traveling on parallel roads), and thus, should belong to the same cluster. Therefore, we need a high quality trajectory distance measure that can correctly capture the complex characteristics of mobile object trajectories. Moreover, a distance function for whole trajectories should also be simple enough to support trajectory clustering on a large scale.

In this chapter, we present a methodical approach to clustering whole trajectories of mobile objects traveling in a road network. We propose a three-phase framework, called TraceMob, for clustering whole trajectories. In the first phase, we compute the spatial proximity of whole trajectories by employing $\alpha \times \beta$ grid abstraction over the raw trajectory datasets. We customize TraceMob clustering algorithms by tuning the size of the $\alpha \times \beta$ grid cells to handle both metropolitan (dense) and suburban

(sparse) road networks with consistent and high clustering quality at scale. This development allows TraceMob to correctly cluster trajectories that are in parallel within certain spatial vicinity most of the time regardless of whether they have any overlapping road segments, and successfully separate trajectories that are far away from one another most of the time into different clusters even though they may share some road segments.

In the second phase of TraceMob, we develop *TrajMap* to transform trajectories represented in a road network space into d-dimensional data points in an Euclidean space. We tune the dimensionality d to ensure that trajectories that are within certain spatial proximity in the road network will be mapped to data points that are close in d dimensional Euclidean space and trajectories that represent very different motion behaviors will be mapped to data points that are relatively far away from one another in d-dimensional Euclidean space. One of the important features of our *TrajMap* development is to provide consistent and fair treatment of trajectories of varying sizes during clustering analysis.

In the third phase of TraceMob, we employ a whole trajectory clustering validation model, consisting of an extensible set of clustering quality measures, to validate the quality of clustering analysis in both road network space and transformed d-dimensional Euclidean space. We allow users to iteratively execute the three phase clustering analysis of TraceMob by adjusting the settings of $\alpha \times \beta$ cell size, the dimensionality d and the number of preferred clusters K during the abstraction, transformation and validation process. We conduct extensive evaluation on TraceMob using mobile traces generated on real road network maps of different sizes and density skewness. Experimental results show that TraceMob effectively discovers the cluster structure of road-network trajectory datasets and runs several orders of magnitude faster than the existing whole trajectory clustering algorithms.

The rest of the chapter proceeds as follows. Section 3.2 gives a brief overview of

related work. Section 3.3 presents an overview of TraceMob three phase clustering analysis framework. We provide an in-depth description for each of the three phases in Section 3.4, Section 3.5 and Section 3.6 respectively. We report our experimental results in Section 3.8 and conclude the chapter in Section 3.9.

3.2 Related Work

The TraceMob development is related to and inspired by the research in grid indexing, traditional data clustering in general and trajectory clustering in particular.

Grid indexing. The notion of a grid partition of the search space was first introduced with the grid file [68], where a grid structure is used to provide multikey access to files. The grid file is designed to manage multiple attribute records, where each attribute domain is partitioned into disjoint intervals to create the grid file structure. It was shown to be well suited for record inserts/deletes/updates and efficient processing of range queries.

For location-based services, existing work [27,65,87] also use the grid structure to support range and k-NN queries in memory which were shown to be more efficient than the traditional R-tree based solutions. These methods aim at managing individual positions of mobile objects, not their moving paths as a whole. To the best of our knowledge, we are the first to use the grid structure of a road network to represent and compute the trajectory dissimilarity with respect to clustering in trajectory-based services. Compared to the widely used location point-based representation and segment-based representation of trajectories, our grid-based representation offers better performance at reducing the data space as well as capturing the spatial correlation of trajectories.

Traditional Data clustering. Clustering is an important data mining technique to organize data into group of similar objects. Many data clustering algorithms have

been proposed in the literature. Different clustering paradigms use different definitions and evaluation criteria. Basic data clustering techniques includes partitioning and hierarchical approaches. Partitioning clustering partitions the data into a certain number of groups and iteratively refines those groups to optimize a clustering criterion (e.g., minimizing the sum of squared errors within each cluster as in k-means algorithm [62]). The iterative process stops when the optimum cluster quality with respect to that criterion has been reached. K-means and its derivation including PAM [53], CLARA [52], CLARANS [67] are the representatives for this class of clustering techniques. Hierarchical clustering algorithms can either start with one cluster of the total objects (divisive) or with a set of clusters of each individual object (agglomerative) to build a cluster hierarchy. A stop condition is required to terminate the algorithm. The final level in the cluster hierarchy where it reaches the stop condition will be the clustering results. BIRCH [98] and CURE [38] are the widely used hierarchical agglomerative methods. Besides the two main clustering schemes, there are other classes of clustering techniques which are density-based, grid-based, graph-based and model-based approaches. In density-based methods such as DBSCAN [31] and OPTICS [13], clusters are defined as region of high density. An object is a member of a cluster if it has at least $MinPts$ neighboring objects within a given radius ϵ. All the objects in its ϵ-neighborhood are also members of the same cluster. Otherwise the object is classified as noise.

Although various clustering techniques have been introduced over the years, there is still no general technique that is the best to solve the clustering problem for all types of datasets. One method may perform well for a specific dataset while it performs poorly on other datasets because diverse types of data and different objectives of data analysis applications need different clustering approaches.

Trajectory clustering. Research in trajectory clustering to date can be classified

into three categories based on whether data points, subtrajectories or whole trajectories are used as the units of clustering.

Point-based approaches use each position point in trajectories as the clustering unit and cluster a trajectory dataset by transforming trajectories, the time series of location points, into a large location position dataset, and then applying traditional clustering algorithms to cluster position based location dataset instead of directly clustering whole trajectories [21, 95, 99]. Clustering results in this category can help identify hot spots but fail to discover subtrajectory or whole trajectory patterns.

Subtrajectory based approaches use subtrajectories as the unit of clustering and aim at clustering sub-trajectories of the whole trajectories to discover interesting subtrajectory clusters. In these clustering schemes, trajectories are first split into trajectory fragments [36, 40, 56], which are used as the clustering units. The unique movement of a mobile object is ignored since different subtrajectories of a whole trajectory may belong to different sub-trajectory clusters.

In contrast with the first two categories, the third category treats whole trajectories as the clustering units. Whole trajectory clustering presents a challenging technical issue: how to measure the distance between two whole trajectories without bias of long or short trajectories and taking into account approximately parallel trajectories. The approaches [24, 25] using Minkowski and Edit Distance only work on trajectories of equal size. To deal with trajectories of varying sizes, several approaches are proposed. [55, 57, 94] use dynamic time warping distance to stretch trajectories by repeating their coordinate values, then compute the Euclidean distance on the stretched trajectories. These measures have high computational complexity due to the costly L_p-norm computation and are very sensitive to noise. LCSS [86] and its variants are based on subsequence matching instead of complete sequence matching like aforementioned approaches. [33] uses a mixture model based clustering algorithm with regression components as trajectory clusters. Expectation-Maximization (EM)

is used to estimate which component each trajectory belongs to. However, this approach is only suitable for short trajectories, such as gene trajectories or trajectories that can be expressed as a function of time in order to use regression based transformation. Another approach [74] adapts the traditional hierarchical agglomerative clustering algorithm. The trajectory distance is measured by computing the shortest path distance between every pair of vertices from two trajectories, which takes $O(nlogn+m)$ for a network of n vertices and m edges. Clearly, this approach has very high computational cost and fails to scale to clustering large trajectory datasets. The work in [91] improves the computational complexity by using a simple distance measure based on the number of shared road segments between two whole trajectories and then employs FastMap [32] to transform trajectory datasets to high-dimensional data points. However, this approach fails to handle some common cases: (i) trajectories that have no common road segments but are in parallel within certain spatial vicinity most of time, (ii) trajectories that belong to distinct clusters though they have some shared road segments, and (iii) segment based distance measure tends to bias long trajectories over short trajectories. Also, there is no tuning of dimensionality to preserve the trajectory distance in the transformed multidimensional Euclidean space. Furthermore, existing distance measurements in whole trajectory clustering examine all the recorded positions in a trajectory, which is inefficient and often unnecessary for road network trajectories.

With these problems in mind, TraceMob by design aims at meeting the dual objectives of whole trajectory clustering: the clustering distance measure should capture the complex spatial characteristics of trajectories in road networks and yet simple enough to support trajectory clustering on a large scale.

3.3 Overview

In this section, we first present the reference model of road networks and grid overlay of a road network. Then we give a brief overview of TraceMob three phase framework and system architecture for whole trajectory clustering.

3.3.1 Road Network Trajectories

A road network is modeled by a single directed graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_0, v_1, \ldots, v_N\}$ is a set of road intersections and $\mathcal{E} = \{(sid, v_i v_j) | v_i, v_j \in \mathcal{V}\}$ is a set of directed edges connecting the road intersections.

Each edge $e = (sid, v_i v_j) \in \mathcal{E}$ is identified by the road segment id sid which connects two road intersections v_i and v_j in the real road network. The length of a road segment $e = (sid, v_i v_j)$ is denoted by $len(e)$.

We define a *road network location* as a tuple of three elements $l = (sid, (x, y), t)$, where sid is the identifier of road segment where the object resides, (x, y) is the geometric coordinates of the object's location, and t is the timestamp when the location is recorded.

A road network trajectory Tr, denoted by $Tr_{=}(trid, l_1 l_2 ... l_L)$, is a time-ordered sequence of road network locations of length L and is uniquely identified by a trajectory identifier $trid$.

3.3.2 Grid Structure

Given a road network G, we refer to the minimum bounding rectangle (MBR) region that covers the entire road network as the *universe of discourse*, defined by $U(G) = Rect(X, Y, W, H)$, where X is the x-coordinate and Y is the y-coordinate of the lower left corner of the MBR corresponding to the universe of discourse. W is the width and H is the height of the universe of discourse. X, Y, W and H are system parameters to be set at the system initialization time.

Grid and Grid cells. We define a grid overlay of the road network G by partitioning $U(G)$ into a grid of contiguous rectangular cells of size $\alpha \times \beta$. Formally, we define the grid overlay of G as follows: $\mathcal{A}_{Grid(G)}(U(G), \alpha, \beta) = \{A_{i,j} : 1 \leq i \leq M, 1 \leq j \leq N, A_{i,j} = Rect(X+i*\alpha, Y+j*\beta, \alpha, \beta), M = \lceil H/\alpha \rceil, N = \lceil W/\beta \rceil\}$. α and β are system parameters that define the cell size of the grid. $A_{i,j}$ is an $\alpha \times \beta$ rectangle area representing the grid cell that is located on the ith row and jth column of the grid $\mathcal{A}_{Grid(G)}$. Each cell entry $A_{i,j}$ is uniquely identified by its cell identifier cid, which is an integer in $[1, M * N]$.

$$A_{i,j}.cid = (i-1) * N + j, i \in [1, M], j \in [1, N] \tag{10}$$

Position to Grid Cell Mapping. Let $l.pos$ denote the geometric coordinates (x, y) of a road network location $l = (sid, (x, y), t)$. Let $A_{i,j}$ denote a cell in the grid $\mathcal{A}_{Grid(G)}$. $Pmap(l.pos)$ is a road network location to grid cell mapping, defined as $Pmap(l.pos) = A_{\lceil \frac{pos.x-X}{\alpha} \rceil, \lceil \frac{pos.y-Y}{\beta} \rceil}$.

Adjacent cells. Given a grid cell $A_{i,j}$ in a grid $\mathcal{A}_{Grid(G)}$, we use $adjacent\_cells(A_{i,j})$ to denote the set of grid cells which share one edge with $A_{i,j}$.

Thus, each cell $A_{i,j}$ has four adjacent cells when $1 < i < M, 1 < j < N$, and $adjacent\_cells(A_{i,j}) = \{A_{i-1,j}, A_{i+1,j}, A_{i,j-1}, A_{i,j+1}\}$ $(1 < i < M, 1 < j < N)$. When $i = 1$ or $i = M$ and $1 < j < N$, or $1 < i < M$, $j = 1$ or $j = N$, $A_{i,j}$ has three adjacent cells. When $i = j = 1$ or $i = M, j = N$, $A_{i,j}$ has two adjacent cells.

Cell to subtrajectory mapping. Let $A_{i,j}$ denote a grid cell, $Tr = (trid, l_1 l_2 ... l_L)$ denote a trajectory and $l_{i_1} \ldots l_{i_m}$ be a subsequence of $l_1 l_2 ... l_L$, $i_1, \ldots, i_m \in [1, L]$. $CTmap(A_{i,j}, Tr)$ is a grid cell to trajectory mapping which returns the subtrajectory contained by the grid cell, defined by $CTmap(A_{i,j}, Tr) = l_{i_1} \ldots l_{i_m}$ such that $A_{i,j}.cid = Pmap(l_{i_1}.pos).cid = \cdots = Pmap(l_{i_m}.pos).cid$.

For presentation convenience, all the definitions in the rest of the chapter are assumed to be given in the context of a road network $G = (\mathcal{V}, \mathcal{E})$ and its grid overlay

$\mathcal{A}_{Grid(G)}$.

3.3.3 Design Consideration and Clustering Framework

In this section, we give a brief overview of TraceMob with respect to its design consideration and its framework for whole trajectory clustering.

Trajectories collected via GPS sensing are often quite long in terms of the number of the recorded locations, especially with frequent periodic sensing interval. Let $Tr = (trid, l_1 l_2 ... l_L)$ $(1 \leq L)$ denote a trajectory and each trajectory location l_i consists of the road segment sid and coordinate of (x, y). Long trajectories are those consisting of large number of location points and thus large L value. By utilizing road network characteristics, we can abstract road network trajectories using road segments without loss of data quality. Thus, a trajectory $Tr = (trid, l_1 l_2 ... l_L)$ can be alternatively represented by its road segment sequence, denoted by $Tr_{\mathcal{E}} = (trid, e_1 e_2 ... e_R)$ $(e_i \in \mathcal{E}, 1 \leq i \leq R, R \leq min(L, |\mathcal{E}|))$. Similarly, the length of a trajectory Tr can be represented by either the number of location samples (L) or the actual length of the trajectory in miles or kilometers by summation of the lengths of all the road segments associated to the trajectory, namely $len(Tr) = len(e_1) + len(e_2) + ... + len(e_R)$.

An intuitive but naïve approach to compute the distance between two trajectories is to measure the level of overlapping between the two trajectories by the length of their common road segments. We refer to such a road segment based distance metric as SegSD. Let $len(Tr_i \cap_{\mathcal{E}} Tr_j)$ denote the length of common road segments of two trajectories Tr_i, Tr_j, the distance between Tr_i and Tr_j can be computed as follows:

$$SegSD = 1 - \frac{len(Tr_i \cap_{\mathcal{E}} Tr_j)}{len(Tr_i) + len(Tr_j) - len(Tr_i \cap_{\mathcal{E}} Tr_j)} \tag{11}$$

Instead of using the number of common road segments, this distance function (using normalized distance) is less sensitive to the number of road segments and is effective in capturing overlapping level between any pair of trajectories. By design, the more common road segments and the longer the two trajectories are, the smaller the SegSD

distance value will be.

However, the SegSD metric has several inherent drawbacks. First, when the two trajectories do not have any road segment in common, SegSD will return the highest distance value of 1. Thus by SegSD, trajectories that have no common road segments even though are close to one another most of time in terms of spatial vicinity, such as two trajectories representing two traveling paths on nearby parallel roads, will be mistakenly treated as the most far away and most dissimilar. For example, in Figure 18(c), trajectories Tr_1 and Tr_2 are located on the same road segments. Trajectory Tr_3 is very close to both Tr_1 and Tr_2 but is located on different road segments. Using the segment-based distance function Tr_1 and Tr_2 are clustered together but Tr_3 will be incorrectly clustered into a separate cluster.

Second, if two trajectories representing two completely different trajectory clusters (such as one represents traveling path from east to west of Atlanta and another represents traveling path from south to north of Atlanta), but they have one road segment in common, then these two trajectories will have smaller SegSD value than the highest SegSD distance value of 1. Thus, they are treated as closer than the two trajectories that are in parallel within close spatial vicinity most of time and both from south to north with destination in Alpharetta, though they have no common road segment (e.g., one traveling on I-85 north and the other traveling to Alpharetta on local roads).

Third but not the least, by SegSD, there is no distinction between two trajectories that share a subsequence of road segments and two trajectories that share disconnected common road segments if the total actual distance of the shared segments is the same. However, in reality, the former should be considered closer in terms of spatial distance than the latter.

To address these common problems inherent in the simple segment based distance metric, we introduce a grid based distance metric in TraceMob to measure the

spatial distance between two whole trajectories. In the next section we will describe how TraceMob capitalizes on grid structure [68] to provide simple and customizable abstraction of spatial closeness of trajectories.

Given that users may have personalized criteria on clustering quality, in TraceMob we support both distance metrics and suggest the grid based distance metric over the segment based distance metric if the users are more sensitive to the above problems and their impact on clustering quality.

In addition to the grid based trajectory distance metric, TraceMob employs a three-phase framework for whole trajectory clustering. Prior to the three-phase clustering analysis, TraceMob initializes the system through a number of system-supplied configuration parameters (such as $\alpha \times \beta$ cell size) and user-defined parameters (such as the preferred distance measure, the number of clusters). In ***Phase I***, the raw trajectory dataset is processed according to the selected distance metric for computing distances between each pair of whole trajectories. In ***Phase II***, TrajMap is employed to transform each trajectory into a d-dimensional point in an Euclidean metric space. d is determined to ensure that the distance between any pair of trajectories in a road network is best preserved by the distance between the corresponding pair of points in the d dimensional metric space. In ***Phase III***, we employ k-means clustering with user-defined K and the optimal initial centroids [14], followed by clustering quality validation. Our three phase clustering process for trajectory abstraction, transformation and clustering with validation can be executed iteratively and users can adjust the setting of some parameters, such as the dimensionality d, the number of preferred clusters K, to obtain customized clustering result. In the subsequent three sections, we will describe the technical details for each of the three phases.

We will use the grid cells as the building blocks for the TraceMob clustering framework. Though in this chapter we use a grid topology with cells of equal size for simplicity, our methods can be easily extended to grids with cells of different shapes

and/or sizes, such as an adaptive grid represented by a quadtree structure.

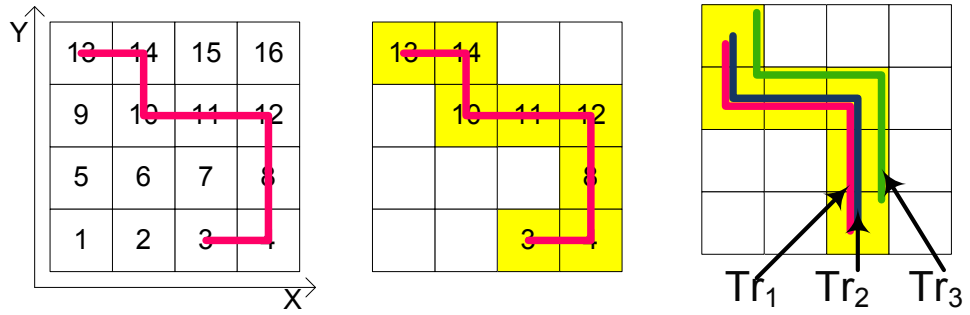
3.4 Trajectory Distance Measures

We have briefly introduced segment based distance metric and the three-phase framework for whole trajectory clustering in the previous section. This section is dedicated to describe the technical development of Phase I. We first describe the grid-based representation of road network trajectories and define the concept of trajectory overlapping and introduce the Simple Grid-Based Distance metric (SGBD) as an intuitive baseline distance function. Then we analyze the weaknesses inherent in SGBD and introduce a number of key concepts, such as overlapping cell sequences, mergeable cells, proximity based trajectory intersection and union, to capture the complex spatial correlations between trajectories. We introduce the Grid Cell Sequence Distance (GridCSD) function as the recommended grid-based distance metric to measure the pairwise spatial proximity of trajectories.

3.4.1 Simple Grid-Based Distance Function (SGBD)

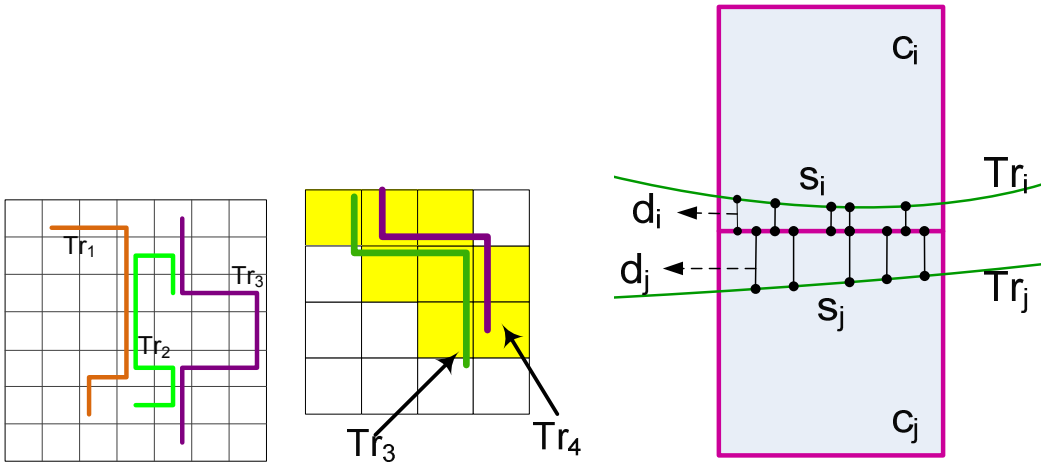
In TraceMob, each trajectory and its road network locations are indexed by the grid matrix $\mathcal{A}_{Grid(G)}$. Thus, a trajectory $Tr = (trid, l_1 l_2 ... l_L)$ can be represented by a time ordered sequence of grid cells covering its locations, denoted by $Tr_{\mathcal{A}} = (trid, c_1 c_2 ... c_n)$, where each $c_k (1 \leq k \leq n)$ is a grid cell covering a subsequence of $l_1 l_2 ... l_L$, denoted by $l_{i_1} \ldots l_{i_m}$. Figures 18(a) and 18(b) illustrate the grid-based representation of a trajectory. Given a uniform grid of size 4×4, we have 16 grid cells, each with a cell identifier. A trajectory depicted as a polyline in Figure 18(a) will be represented by a sequence of highlighted cells $\{13, 14, 10, 11, 12, 8, 4, 3\}$ covering the trajectory path as shown in Figure 18(b).

Let $sub_{\mathcal{A}}(c_k, Tr)$ denote the function that extracts the subsequence $l_{i_1} \ldots l_{i_m}$ residing in grid cell c_k from Tr, $|Tr_{\mathcal{A}}|$ denote the grid cell cardinality of Tr, i.e., the number of grid cells in the grid cell sequence $Tr_{\mathcal{A}}$, and $Tcells(Tr)$ denotes the set



(a) (b) (c)

Figure 18: Examples of grid-based representation and overlapping cells



(a) (b) (c)

Figure 19: Overlapping cell sequences and mergeable cells

of grid cells that the trajectory Tr is passing through. We can define the concept of trajectory overlapping as follows.

Definition 12 *(Trajectory Overlapping)* Trajectory overlapping is defined over two road network trajectories Tr_i and Tr_j, denoted by $share(Tr_i, Tr_j)$, which consists of the set of common grid cells shared by their grid-based representation $Tr_{i\mathcal{A}}$ and $Tr_{j\mathcal{A}}$. Formally, $share(Tr_i, Tr_j) = Tcells(Tr_i) \cap Tcells(Tr_j)$.

Let $|share(Tr_i, Tr_j)|$ denote the cardinality of $share(Tr_i, Tr_j)$, the number of common cells shared by the two trajectories. Using trajectory overlapping $share(Tr_i, Tr_j)$, we devise a distance function, called ***Simple Grid-Based Distance Function***

(SGBD), which computes the distance of two trajectories, Tr_i and Tr_j, as follows:

$$SGBD = 1 - \frac{|share(Tr_i, Tr_j)|}{|Tr_{i\mathcal{A}}| + |Tr_{j\mathcal{A}}| - |share(Tr_i, Tr_j)|} \tag{12}$$

The SGBD distance function improves the segment based distance metric SegSD in terms of exploiting grouping opportunities for non-overlapping trajectories. For example, when trajectories do not share any common road segments but are very close to each other, the grid-based distance measure can more accurately capture the spatial distance of such trajectories than the road-segment based distance metric. Consider the example in Figure 18(c) all three trajectories are represented using the same sequence of grid cells. Thus, the grid-based abstraction and distance function can better reflect the spatial proximity of all three trajectories compared to the road-segment based distance function.

However, SGBD treats common cells shared between two trajectories as independent cells. Thus it fails to take into account the continuity of the overlapping cells in the trajectory distance measure. It also does not consider the trajectories that are in adjacent cells but are within close vicinity. Consider Figure 19(b), although covered by separate sequences of cells, trajectories Tr_3 and Tr_4 are close to each other.

To address these problems, we introduce the second grid based trajectory distance metric as the recommended trajectory distance metric in TraceMob.

3.4.2 Grid Cell Sequence Distance Function (GridCSD)

We first introduce the concept of *overlapping cell sequences* and the concept of *mergeable cells* and then define the GridCSD trajectory distance function.

Definition 13 *(Overlapping Cell Sequences)*

Let Tr_i and Tr_j denote two road network trajectories having $|share(Tr_i, Tr_j)| > 0$, $Tr_{i\mathcal{A}}$ denote the sequence of grid cells covering Tr_i, and $\in_{sub}$ denote the subsequence relation. We say that $ocs(Tr_i, Tr_j) = \{ocs_1, ocs_2, ...ocs_m\}$ is an ordered list of overlapping cell sequences extracted from $share(Tr_i, Tr_j)$ if and only if $ocs(Tr_i, Tr_j)$

satisfies the following conditions: (i) For each cell sequence $ocs_k \in ocs(Tr_i, Tr_j)$: $ocs_k \in_{sub} Tr_{i\mathcal{A}}$ and $ocs_k \in_{sub} Tr_{j\mathcal{A}}$. We use $cells(ocs_k)$ to denote the set of grid cells of ocs_k. (ii) Two consecutive cell sequences $ocs_k, ocs_{k+1} \in ocs(Tr_i, Tr_j)$ are not continuous in $Tr_{i\mathcal{A}}$ and in $Tr_{j\mathcal{A}}$. (iii) $share(Tr_i, Tr_j) = \cup_{k=1}^{m} cells(ocs_k)$.
We use $|ocs(Tr_i, Tr_j)|$ to denote the number of cell sequences in $ocs(Tr_i, Tr_j)$.

Clearly, two trajectories are closer in distance not only when they share more common grid cells, but they may also have less number of overlapping sequences. This is especially true when two pairs of trajectories have the same number of common cells but one has less overlapping sequence than the other showing that their two mobile objects are more similar in their movement. For example, in Figure 19(a), we have $|share(Tr1, Tr2)| = |share(Tr2, Tr3)| = 4$ and $|ocs(Tr1, Tr2)| = 1 < |ocs(Tr2, Tr3)| = 2$. Intuitively, Tr_2 is closer to Tr_1 than to Tr_3.

Definition 14 *(Mergeable Cells)* Given two trajectories Tr_i, Tr_j and a distance threshold δ, two grid cells c_i and c_j are *mergeable* with respect to Tr_i and Tr_j if (i) $c_i \in Tcells(Tr_{i\mathcal{A}}) \backslash share(Tr_i, Tr_j)$ and $c_j \in Tcells(Tr_{j\mathcal{A}}) \backslash share(Tr_i, Tr_j)$; (ii) c_i and c_j are adjacent cells, namely $c_i \in adjacent\_cells(c_j)$ and $c_j \in adjacent\_cells(c_i)$; (iii) two subsequences $s_i = l_{i_1} l_{i_2} ... l_{i_m} = sub_{\mathcal{A}}(c_i, Tr_i)$ and $s_j = l_{j_1} l_{j_2} ... l_{j_n} = sub_{\mathcal{A}}(c_j, Tr_j)$ are satisfying:

$$\mathcal{L}(s_i, s_j) = \frac{\sum_{k=0}^{m} d_{ik}^2}{\sum_{k=0}^{m} d_{ik}} + \frac{\sum_{q=0}^{n} d_{jq}^2}{\sum_{q=0}^{n} d_{jq}} < \delta \tag{13}$$

Where d_{ik} and d_{jq} are the perpendicular distances from locations in subsequences s_i and s_j to the common edge of c_i and c_j respectively, as illustrated in Figure 19(c) The left side of the inequality in Formula 13 measures the sum of the Lehmer mean L_p $(p = 2)$ of these perpendicular distances.

We use $merge(Tr_i, Tr_j)$ as a function to compute the set of meargeable cell pairs from two trajectories Tr_i and Tr_j, which is described in Algorithm 4. After computing the set of common grid cells $share(Tr_i, Tr_j)$ (line 2), the remaining cells in each

Algorithm 4 merge(Tr_i, Tr_j)

Input: Trajectories Tr_i, Tr_j and distance threshold δ
Output: a set $\mathcal{M}$ of mergeable cells

```
1: M = ∅
2: compute share(Tr_i, Tr_j)
3: list_i ← Tmap(Tr_iA)\share(Tr_i, Tr_j)
4: list_j ← Tmap(Tr_jA)\share(Tr_i, Tr_j)
5: for c_i ∈ l_i do
6:    for c_j ∈ l_j do
7:       if c_i and c_j are adjacent then
8:          s_i ← sub_A(c_i, Tr_i); s_j ← sub_A(c_j, Tr_j)
9:          if L(s_i, s_j) < δ then
10:            add the cell pair (c_i, c_j) to M
11:            remove c_i from list_i and c_j from list_j
12:         end if
13:      end if
14:   end for
15: end for
16: return M
```

trajectory (lines 3-4) are scanned to detect pairs of mergable cells (lines 5-15). After computing the set of common grid cells $share(Tr_i, Tr_j)$, the remaining cells in each trajectory are scanned to detect pairs of mergable cells. We denote $merge(Tr_i, Tr_j)$ as a function to compute the set of meargeable cell pairs from two trajectories Tr_i and Tr_j. The number of cell pairs in $merge(Tr_i, Tr_j)$ is denoted by $|merge(Tr_i, Tr_j)|$.

The purpose of *mergeable cells* is to capture the proximity of parts of trajectories residing in adjacent cells but are still close to each other with respect to a distance threshold δ. Note that we can set δ to $min(\alpha, \beta, \sqrt{\alpha \times \beta})$ since each cell area defines a measure of the spatial proximity in the road network space. When we compute the intersection of two trajectories, we consider both their common cells and mergeable cells.

Definition 15 *(Proximity-Based Intersection)*

The proximity based intersection of two road network trajectories Tr_i and Tr_j, denoted by $Tr_i \cap_{\mathcal{A}} Tr_j$, consists of the grid cells in $share(Tr_i, Tr_j)$ or in $merge(Tr_i, Tr_j)$.

Its cell cardinality is computed as follows:

$$|Tr_i \cap_{\mathcal{A}} Tr_j| = |share(Tr_i, Tr_j)| + |merge(Tr_i, Tr_j)| \tag{14}$$

Definition 16 *(Proximity-Based Union)*

The proximity based union of two road network trajectories Tr_i and Tr_j, denoted by $Tr_i \cup_{\mathcal{A}} Tr_j$, consists of the grid cells in $Tr_{i\mathcal{A}}$ or $Tr_{j\mathcal{A}}$ but not in $Tr_i \cap_{\mathcal{A}} Tr_j$. Its cell cardinality is computed as follows:

$$|Tr_i \cup_{\mathcal{A}} Tr_j| = |Tr_{i\mathcal{A}}| + |Tr_{j\mathcal{A}}| - |Tr_i \cap_{\mathcal{A}} Tr_j| \tag{15}$$

Based on the proximity-based intersection and union of two trajectories, we formally define the **Grid Cell Sequence Distance** function - GridCSD, to measure the pairwise distance of trajectories in the road network.

Definition 17 ***(GridCSD)*** The distance of two road network trajectories Tr_i and Tr_j is measured by the distance of their grid cell sequences, denoted by $GridCSD(Tr_i, Tr_j)$, which is computed from their proximity-based intersection and union as follows:

$$GridCSD(Tr_i, Tr_j) = 1 - w \times \frac{|Tr_i \cap_{\mathcal{A}} Tr_j|}{|Tr_i \cup_{\mathcal{A}} Tr_j|} \tag{16}$$

Where $w = 1/|ocs(Tr_i, Tr_j)|$ if $|ocs(Tr_i, Tr_j)| > 0$, otherwise $w = 1$.

GridCSD in Formula 16 by design takes into account both the overlapping grid cells and mergeable grid cells from two trajectories and are in favor of longer common cell sequences. By GridCSD, two trajectories have smaller distance value if they share more common cells or mergeable cells or have a smaller number of overlapping cell sequences. The weight $w = 1/|ocs(Tr_i, Tr_j)|$ in Formula 16 is to ensure that if two pairs of trajectories share the same set of cells, then the pair of trajectories that share longer overlapping cell sequences (i.e., less number of overlapping cell sequences) will have smaller GridCSD distance.

Compared to SegSD and SGBD, GridCSD is more advanced and it works for both trajectories that share common portions and trajectories that have no overlapping segments or no overlapping cells but are still close to one another in terms of spatial trajectory proximity. Therefore, GridCSD captures the spatial proximity of trajectories more accurately than segSD (segment based distance metric) and SGBD (simple grid cell based distance metric).

Choice of Cell Size. The grid cell size defined by α and β is tunable in TraceMob. For example, for suburban area, the road network is sparse and thus the grid cell size can be relatively larger compared to the grid cell size for road networks of metropolitan cities. in the first prototype of TraceMob, we let system administrator or end user to set the lower and upper bounds of the segment count per grid cell, denoted by n_l and n_u. Given that segments are not uniformly distributed across a road network, the average number of road segments per cell should be within an appropriate range defined by $[n_l, n_u]$. Thus α and β should satisfy the following condition: $n_l \leq \lfloor \frac{|\mathcal{E}| \times \alpha \times \beta}{H \times W} \rfloor \leq n_u$, where $|\mathcal{E}|$ is the total number of road segments in the road network $G = (\mathcal{V}, \mathcal{E})$ with H and W as the height and width of the MBR of G.

Complexity Analysis. All three distance functions GridCSD, SGBD and SegSD are based on Jaccard coefficient formula over sets, which takes the size of the intersection of two sets divided by the size of their union, and are normalized between 0 and 1. They need to scan the two cell sequences (for GridCSD and SGBD) or two segment sequences (for SegSD) to compute the overlapping grid cells or road segments. Therefore, SGBD takes $\mathcal{O}(|Tr_{i\mathcal{A}}| + |Tr_{j\mathcal{A}}|)$. GridCSD takes $\mathcal{O}(3 * (|Tr_{i\mathcal{A}}| + |Tr_{j\mathcal{A}}|))$ since it needs to scan for meargable cells and continuous common grid cells. Let $Tseg(Tr)$ denotes the set of road segments covered by trajectory Tr. SegSD takes $\mathcal{O}(|Tseg(Tr_i)| + |Tseg(Tr_j)|)$.

3.5 Trajectory Mapping

In this section we present the technical development of Phase II of our whole trajectory clustering − Mapping each trajectory into a d-dimensional data point. This transformation enables TraceMob to better optimize the whole trajectory clustering process. If we utilize the pairwise distance values obtained in Phase I to perform k-means algorithm directly over the road network trajectories, we will face two technical challenges. First, we need to choose the k trajectories to serve as the k initial centroids, and the existing statistics based algorithms for selecting the best initial centroids are all developed for Euclidean space [14]. Second, we need to address the problem of clustering validation since most of cluster validation metrics developed to date are designed for evaluating the quality of clustering datasets in Euclidean space. This motivates us to develop *TrajMap* to transform each trajectory into a d-dimensional point in a Euclidean metric space. d is determined to ensure that the distance between any pair of trajectories in a road network is best preserved by the distance between the corresponding pair of points in the d dimensional metric space.

In terms of trajectory mapping, we need to address two key issues: how to select the best initial projection axis and how to determine the best dimensionality d for a given trajectory dataset. In the rest of this section, we first give an overview of the *TrajMap* algorithm and then give an in-depth discussion on these two key challenges in Section 3.5.2 and Section 3.5.3 respectively. After mapping trajectories to d-dimensional image points, we use the partition based approach to clustering, i.e., given an integer k, we want to partition a set of N trajectories into k clusters. We choose to use the traditional k-means algorithm because of its efficiency and simplicity. In addition, in many trajectory-based application scenarios, the values of k are usually determined a priori.

3.5.1 Transforming Trajectories using TrajMap

TrajMap takes three input parameters, a set of N trajectories, the pairwise trajectory distance function $\mathcal{D}$, the desired dimensionality d, and transform them into N points in a d-dimensional space such that the original distances are preserved.

Concretely, *TrajMap* performs the trajectory mapping task in an iterative manner for d iterations. Each of its d iterations includes three components as displayed in Algorithm 5: (1) finds two *pivot trajectories* to form a projection axis (line 8-7) given a distance function, in the first iteration, 5 calls chooseInitialPivots($\mathcal{T}, \mathcal{D}(), rep$) function to select the most accurate pivot trajectories with respect to the road network space (2) projects the trajectories on the projection axis to compute their coordinates (line 8-11), (3) utilizes the new coordinate $x_{iter}(P_i)$ and $x_{iter}(P_j)$ to obtain a new distance function $\mathcal{D}(Tr_i, Tr_j)$ on a hyperplane perpendicular to the projection axis (line 12), which will be used in the next iteration. We also develop a cost function, called $I_{preserve}$, for determining the best d at each iteration, which allows *TrajMap*$((\mathcal{T}, \mathcal{D}(), d_u))$ to take the system supplied upper bound d_u and iteratively tune the d parameter such that the best d^* may be smaller than d_u and has the lowest cost in terms of $I_{preserve}$. The final result of *TrajMap* is a set of N d-dimensional points, which will be the input for Phase III of our clustering framework.

TrajMap is implemented by extending FastMap [32], which is linear on data size N since it requires $O(dN)$ distance computations to complete the transformation. The main extensions include the grid based trajectory distance function, the selection of the initial pivot trajectories considering both trajectory distance and trajectory maximum coverage and the cost function for determining the best d^* at each iteration.

Algorithm 5 $TrajMap(\mathcal{T}, \mathcal{D}(), d)$

Input: (1) A set of N objects $\mathcal{T} = \{Tr_1, Tr_2, ..., Tr_N\}$
(2) Trajectory distance function $\mathcal{D}(Tr_i, Tr_j)$
(3) Number of dimensions d
Output: A set of N points $\mathcal{P} = \{P_1, P_2, ..., P_N\}$ in a d-dimensional space

1: **for** $iter = 1$ to d **do**
2: choose Tr_a and Tr_b which maximizes $\mathcal{D}(Tr_a, Tr_b)$
3: **if** $iter = 1$ **then**
4: $(Tr_a, Tr_b) \leftarrow$ chooseInitialPivots$(\mathcal{T}, \mathcal{D}(), rep)$
5: **end if**
6: $dmax \leftarrow \mathcal{D}(Tr_a, Tr_b)$
7: record the ids of the pivot objects Tr_a and Tr_b
8: **for each** $Tr_i \in \mathcal{O}$ **do**
9: project the objects on the 'line' (Tr_a, Tr_b)
10: compute coordinate x_{iter} of P_i (based on the cosine law for the 'triangle' $Tr_a Tr_i Tr_b$) as follows:
$\mathrm{x}_{iter}(P_i) = \frac{\mathcal{D}(Tr_a,Tr_i)^2 + dmax^2 - \mathcal{D}(Tr_b,Tr_i)^2}{2dmax}$
11: **end for**
12: define a new distance function $\mathcal{D}'(Tr_i, Tr_j)$ to use in the next iteration:

$$\mathcal{D}'_{ij} = \sqrt{\mathcal{D}(Tr_i, Tr_j)^2 - \sum_{t=1}^{iter} (x_t(P_i) - x_t(P_j))^2}$$

13: **if** input dataset is a sample **then**
14: compute $I_{preserve}$
15: **if** $I_{preserve}(iter) \geq I_{preserve}(iter - 1)$ **then**
16: $d^* = iter - 1$
17: **end if**
18: **end if**
19: **end for**

3.5.2 Algorithm to select initial projection axis

It is important to choose appropriate pivot trajectories in the first iteration such that the pivot trajectories are the farthest apart trajectories in the given dataset, since *TrajMap* computes the d-dimensional coordinates of the trajectories' images through projections. A critical decision is whether using the grid-based trajectory distance function alone is sufficent for selecting the pivot trajectories. We observe that by definition of GridCSD, trajectories that have no overlapping cells and no

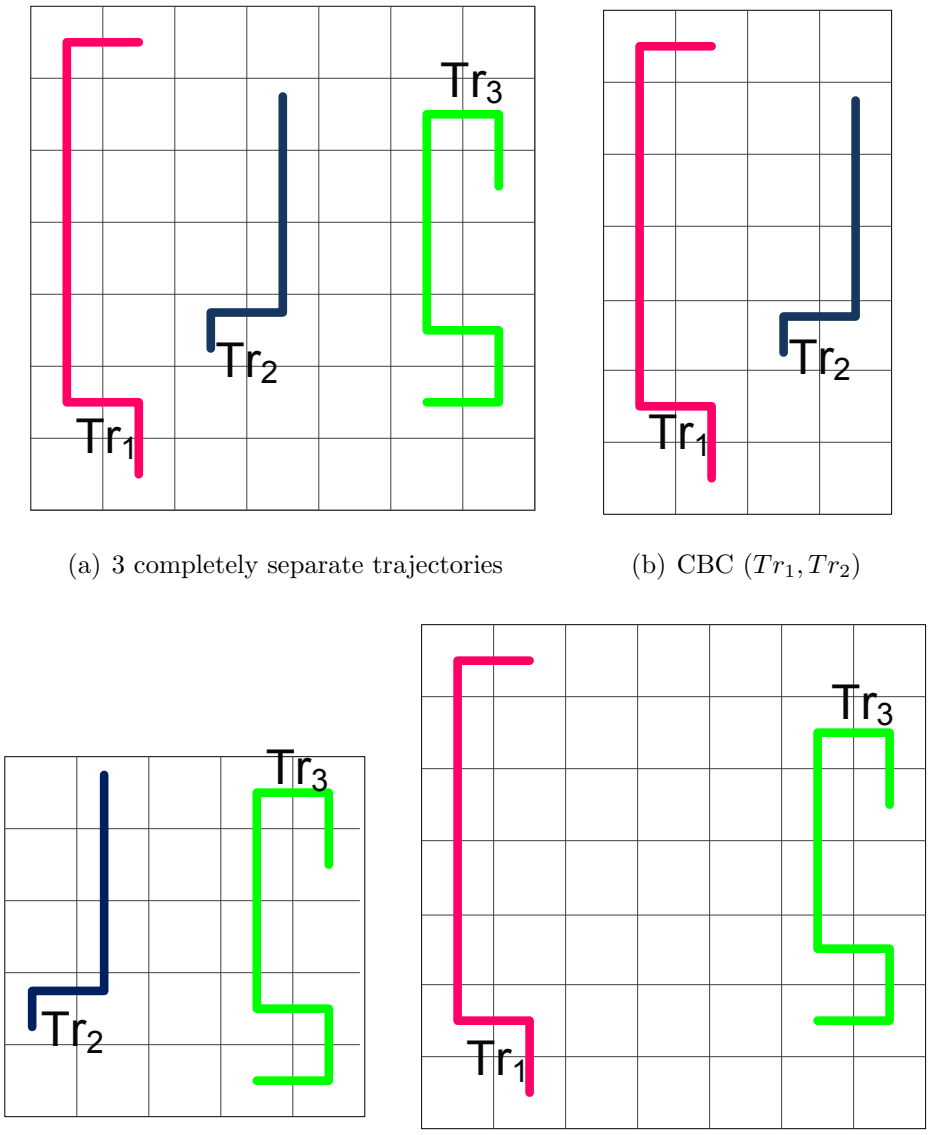
mergeable cells will have the same maximum GridCSD value of 1. Thus, two trajectories have the maximum GridSCD value does not imply that the two trajectories are the farthest apart in a road network. Figure 20 illustrates this observation. We have three trajectories Tr_1, Tr_2 and Tr_3 as displayed in Figure 20(a). Each pair in $\{Tr_1, Tr_2, Tr_3\}$ does not have overlapping cells and mergeable cells so their pairwise distances $GridCSD(Tr_1, Tr_2) = GridCSD(Tr_2, Tr_3) = GridCSD(Tr_3, Tr_1) = 1$. SGBD and SegSD also give three pairwise distance values of 1. However, the bounding box (Figure 20(d)) of Tr_1 and Tr_3 is larger than that of (Tr_1, Tr_2) (Figure 20(b)) and (Tr_2, Tr_3) (Figure 20(c)), which makes Tr_1 and Tr_3 more separate than the other pairs. This motivates us to introduce the *Cell Bounding Coverage* of two trajectories and add the maximum cell bounding coverage condition in choosing the initial pivot trajectories for *TrajMap*.

Definition 18 *(Cell Bounding Coverage)* Given a pair of trajectories Tr_1, Tr_2, their ***Cell Bounding Coverage (CBC)***, denoted by $CBC(Tr_1, Tr_2)$, measures the number of cells in the rectangular region that minimally covers the grid cell sequences representing the given trajectories and is formally computed as follows:

$$CBC = (maxRow - minRow + 1) * (maxCol - minCol + 1) \tag{17}$$

Where $minRow$, $maxRow$, $minCol$ and $maxCol$ are the minimum and maximum values of row index and column index of all the cells in the grid cell sequences $Tr_{1\mathcal{A}}, Tr_{2\mathcal{A}}$. $(minRow, minCol)$ and $(maxRow, maxCol)$ respectively determine the bottom left cell and the top right cell of the rectangular region that forms CBC.

The algorithm for choosing the pivot trajectories first computes the pairwise trajectory distance using GridCSD (or SGBD or SegSD). Then for those pairs of trajectories that have the highest GridCSD value, we compute their CBC values and choose the pair of trajectories with the maximum CBC as the two pivot trajectories in the first *TrajMap* iteration as described in Algorithm 6. Instead of using the number



(a) 3 completely separate trajectories

(b) CBC (Tr_1, Tr_2)

(c) CBC (Tr_2, Tr_3)

(d) CBC (Tr_3, Tr_1)

Figure 20: Example of Cell Bounding Coverage

of repetition (*rep*) as done in Algorithm 6, we can alternatively use a convergence condition such that if the CBC value of the two pivot trajectories computed in the current iteration is similar to the CBC value of the two pivot trajectories computed in the previous iteration by a given threshold γ, then the algorithm terminates.

Algorithm 6 chooseInitialPivots($\mathcal{T}, \mathcal{D}(), rep$)

Input: (1) A set of N trajectories $\mathcal{T} = \{Tr_1, Tr_2, ..., Tr_N\}$
(2) Distance function $\mathcal{D}(Tr_i, Tr_j)$
(3) Number of repetition rep
Output: trajectories Tr_a and Tr_b

1: Randomly pick a trajectory Tr_b
2: **for** $i = 1$ to rep **do**
3: get the set T_A of trajectories which have the maximum trajectory distance to Tr_b, $\mathcal{D}(Tr_i, Tr_b) \geq \mathcal{D}(Tr_i, Tr_j)$, $Tr_i \in T_A$, $Tr_j \in \mathcal{T}_A$
4: choose $Tr_a \in T_A$ which has the maximum $CBC(Tr_a, Tr_b)$
5: get the set T_B of trajectories which have the maximum trajectory distance from Tr_a, $\mathcal{D}(Tr_a, Tr_j) \geq \mathcal{D}(Tr_i, Tr_j)$, $Tr_j \in T_B$, $Tr_i \in \mathcal{T}_B$
6: choose $Tr_b \in T_B$ which has the maximum $CBC(Tr_a, Tr_b)$
7: **end for**
8: return Tr_a and Tr_b as the desired pivot trajectories

3.5.3 Cost Function for Determining d

Intuitively, the larger the number of dimensions d is for the projected trajectory image space, the better *TrajMap* preserves the distances of original trajectories in the d-dimensional trajectory image space. A small d may lead *TrajMap* to have less satisfactory distance preserving quality. But large d may lead to high computational cost due to over-mapping. Therefore, we limit the dimensionality for TrajMap to an upper bound d_u supplied by the system. The actual d will be an integer in $[1, d_u]$ which minimizes the following cost function:

$$I_{preserve} = (I_{stress} + I_{ordering})/2 \tag{18}$$

The cost function $I_{preserve}$ evaluates the distance preservation of TrajMap transformation, which is the average of the *stress index* I_{stress} and the *Kendall tau metric* $I_{ordering}$.

The *stress index* measures the relative difference of the pairwise distances in the

d-dimensional space from the pairwise distances in the original space:

$$I_{stress} = \frac{\sum_{1 \leq i,j \leq N} (\mathcal{D}'_{ij} - \mathcal{D}_{ij})^2}{\sum_{1 \leq i,j \leq N} \mathcal{D}_{ij}^2} \tag{19}$$

where $\mathcal{D}_{ij}$ is the distance between trajectories Tr_i and Tr_j in the road network space and $\mathcal{D}'_{ij}$ is the distance between their images P_i and P_j in the d-dimensional space.

The *Kendall tau metric* [54] measures the ordering dissimilarity of the ordered lists of pairwise distances $\mathcal{L}_{\mathcal{T}}$ and $\mathcal{L}_{\mathcal{I}}$ in the original space and the image space respectively. We have $N(N-1)/2$ pairwise distances in each list. For each pair of distances $(\mathcal{D}_{ij}, \mathcal{D}_{uv})$, if the order of them in $\mathcal{L}_{\mathcal{T}}$ is different from the order of $(\mathcal{D}'_{ij}, \mathcal{D}'_{uv})$ in $\mathcal{L}_{\mathcal{I}}$ then $K(ij, uv) = 1$, otherwise $K(ij, uv) = 0$. The normalized *Kendall tau metric* is computed as follows:

$$I_{ordering} = \frac{\sum_{1 \leq i,j,u,v \leq N} K(ij, uv)}{N(N-1)/2} \tag{20}$$

3.6 Clustering and Cluster Validation

In **Phase III** of TraceMob, we employ k-means clustering over the projected trajectory images in the d-dimensional Euclidean space with user-defined K and the optimal initial centroids [14] followed by clustering quality validation.

To measure the clustering quality, we perform the cluster validation in both the trajectory image space where k-means clustering is performed, and the original road network space where clusters of d-dimensional points are mapped back to clusters of trajectories.

In the d-dimensional space, we evaluate the clustering of N image points $P_1, P_2, ..., P_N$ into a set of k clusters, denoted by $\mathcal{C}_{\mathcal{P}} = (C_1, C_2, ..., C_k)$ using the well known *within cluster sum of squares* (WCSS) of the Euclidean norm $||.||$ which measures the WCSS

distance from a trajectory image point P_j to its assigned cluster center μ_i:

$$wcss = \sum_{C_i \in \mathcal{C}_\mathcal{P}} \sum_{P_j \in C_i} ||P_j - \mu_i||^2 \tag{21}$$

The smaller WCSS value implies the better clustering quality.

After obtaining the clustering result $\mathcal{C}_\mathcal{P} = (C_1, C_2, ..., C_k)$, we replace each point P_j in a cluster C_i with its corresponding trajectory Tr_j and output the final trajectory clusters $\mathcal{C}_\mathcal{G} = (C_1, C_2, ..., C_k)$. We adapt the Silhouette Index [76], a popular method for cluster validation, to measure cluster quality in the road network space.

Definition 19 *(Spatial Silhouette Index)* Given a cluster assignment $\mathcal{C}_\mathcal{G} = (C_1, C_2, ..., C_k)$ of a dataset of N trajectories, the spatial Silhouette Index of $\mathcal{C}_\mathcal{G}$, denoted by SSI, is measured as follows:

$$SSI = \frac{1}{k} \sum_{C_u \in \mathcal{C}_\mathcal{G}} \frac{1}{|C_u|} \sum_{Tr_i \in C_u} \frac{b_i - a_i}{max(a_i, b_i)} \tag{22}$$

Where $a_i = \frac{1}{|C_u|-1} \sum_{Tr_j \in C_u, j \neq i} tf(Tr_i, Tr_j)$,
$b_i = \min_{C_v \in \mathcal{C}_\mathcal{G}, C_v \neq C_u} \frac{1}{|C_v|} \sum_{Tr_j \in C_v} tf(Tr_i, Tr_j)$
and $tf(Tr_i, Tr_j) = CBC(Tr_i, Tr_j)/(|Tr_{iA}| + |Tr_{jA}|)$.

The intuition behind SSI is that for trajectories Tr_i and Tr_j which are grouped together in a cluster, Tr_i and Tr_j are considered *tightly* grouped if their cell bounding coverage tightly wraps around them. We measure this tightness, denoted by $tf(Tr_i, Tr_j)$, using their CBC divided by the sum of their grid cell cardinalities, which is $\frac{CBC(Tr_i, Tr_j)}{|Tr_{iA}|+|Tr_{jA}|}$. The smaller value of $tf(Tr_i, Tr_j)$, the *tighter* Tr_i and Tr_j are. The SSI function computes the local silhouette width ($\frac{b_i - a_i}{max(a_i, b_i)}$) for each trajectory, the average silhouette width for each cluster and the global silhouette width for the total trajectory dataset. Similar to the Silhouette Index, SSI is limited to the interval [-1, 1] and should be maximized.

3.7 TraceMob System Architecture

The first prototype of TraceMob implements the whole trajectory clustering framework with GridSCD, SGBD and SegSD as the distance metrics. Figure 21 shows the sketch of the system architecture. In the trajectory distance computation phase, TraceMob takes as its input a trajectory dataset and a road network $G = (\mathcal{V}, \mathcal{E})$ where the trajectories are recorded or map-matched [89], and uses a grid overlay to abstract a trajectory to a time ordered sequence of grid cells. The output will be a set $\mathcal{T}$ of N trajectories in their grid-based representation and the pairwise distance between each pair of trajectories.

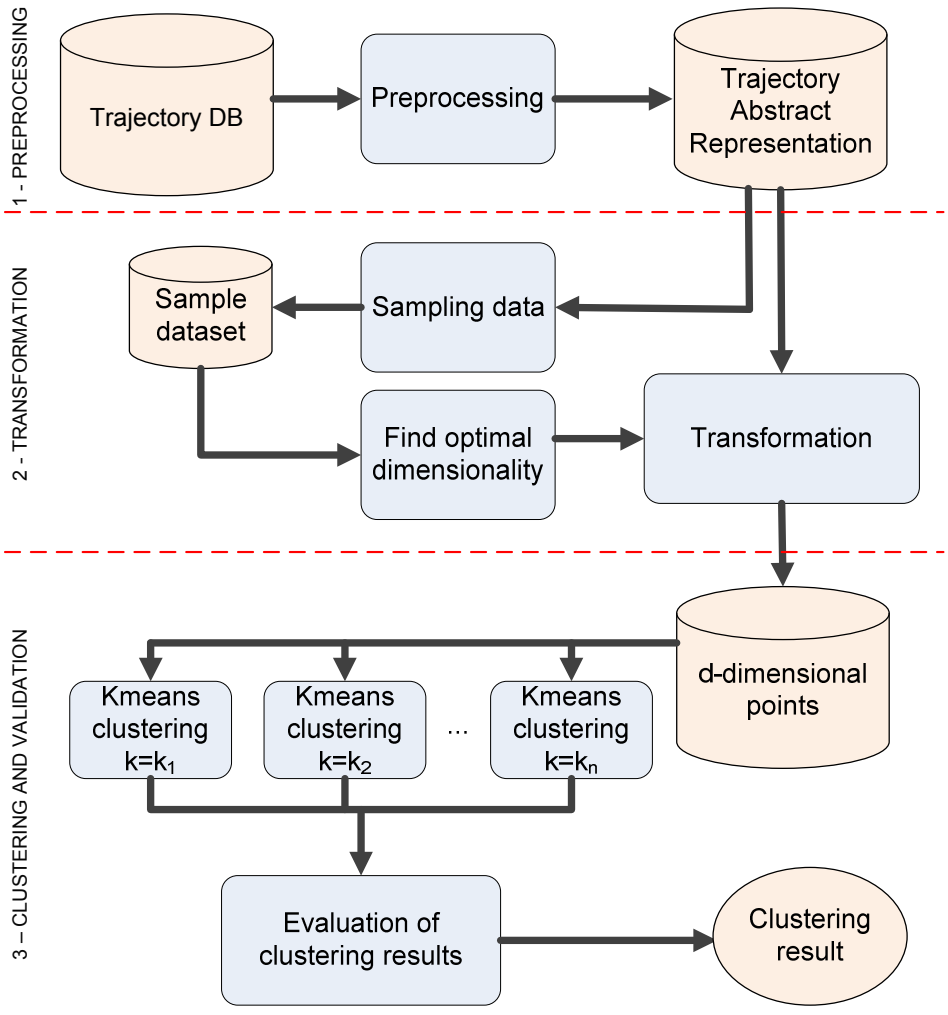


Figure 21: The TraceMob framework.

To determine the grid structure and its matrix representation, a grid cell size needs to be supplied to the trajectory distance computation module. A user can provide their desirable cell size or use the suggested cell size generated by the system. In the former case, the grid size (in number of cells) will be $\lceil \frac{W}{\alpha} \rceil \times \lceil \frac{H}{\beta} \rceil$ given the user input cell size of $\alpha \times \beta$ and $U(G) = Rect(X, Y, W, H)$ - the MBR of the road network G. In the latter case, we use a quad grid of size $2^m \times 2^m$. Given that a cell area defines the spatial proximity in the road network $G = (\mathcal{V}, \mathcal{E})$, to select the best grid cell size for trajectory clustering purpose, we suggest that a grid cell should cover the portions from an appropriate number road segments. For example, if we choose a grid cell to cover an average of 1 to 3 road segments, then m can be determined to satisfy the condition of $1 \leq \lfloor \frac{|\mathcal{E}|}{2^m \times 2^m} \rfloor \leq 3$. Given $U(G) = Rect(X, Y, W, H)$, the grid cell size will be $\lceil \frac{W}{2^m} \rceil \times \lceil \frac{H}{2^m} \rceil$.

In the transformation phase, the *TrajMap* is invoked to map trajectories from their grid-based representation in the road network space to points in a d-dimensional space. To choose the number of dimensions for mapping, TraceMob uses simple random sample $\mathcal{T}_s$ of 10% of the dataset to compute $I_{preserve}$ for each value of d from 1 to $d_u = 50$ by running $TrajMap(\mathcal{T}_s, GridCSD, d_u)$. The value of d^* which minimizes $I_{preserve}$ will be selected. Then TraceMob will execute $TrajMap(\mathcal{T}, GridCSD, d^*)$ to produce a set $\mathcal{P}$ of N d^*-dimensional points.

In the clustering and validation phase, TraceMob performs k-means clustering on $\mathcal{P}$ with one or many values of k supplied by users. The clustering results will then be evaluated in the evaluation phase using our SSI method to output the best clustering result.

In addition to implement the first prototype of TraceMob with the GridCSD distance metric, we also implement TraceMob with SBDS and SegSD for the purpose of performance comparison.

Table 6: The optimal d used in our experiments

Frameworks	**ATL datasets**		**SJ datasets**	
	d^*	$I^*_{preserve}$	d^*	$I^*_{preserve}$
GridCSD-TraceMob	40	0.025	50	0.030
SegSD-TraceMob	41	0.108	40	0.113
SGBD-TraceMob	47	0.071	50	0.078

3.8 Experiments

We perform four sets of experiments to analyze the effectiveness of the TraceMob trajectory clustering framework. We show the effectiveness of using TraceMob with the GridCSD function for preserving the spatial structure of trajectories as well as for producing highly accurate clustering results compared to SGBS-TraceMob and SegSD-TraceMob. Furthermore, we show that the grid based approach runs several orders of magnitude faster than the segment based approach (SegSD-TraceMob) and the direct trajectory clustering without transformation. We implement a k-means clustering with GridCSD using the PAM (Partitioning Around Medoids) algorithm [53] which performs clustering directly on the road network trajectory dataset. We call this implementation GridCSD-PAM.

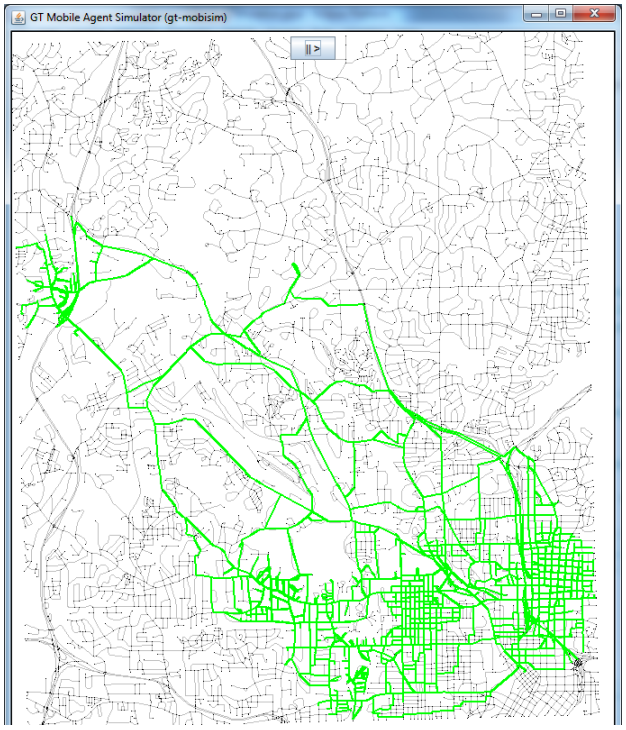
We implement our algorithms using Java and all the experiments are conducted on a PC with Intel Core2 Duo CPU of 2.00GHz and 2GB of main memory.

3.8.1 Datasets and Parameter Settings

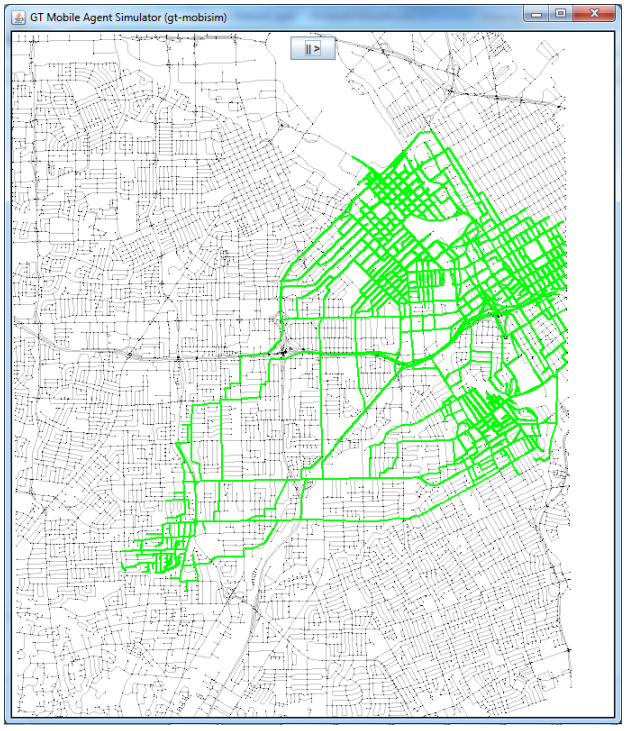
The real user trajectory data are considered highly sensitive regarding location privacy of users. Companies providing LBSs such as Google, Microsoft, Skyhook keep their trajectory data unpublished. Other sources [6, 7] provide sample GPS trajectories but the underlying road networks are unknown and the amount of trajectory data is too small for scalability evaluation. To evaluate the TraceMob with both urban and rural/suburban road networks at different scale of trajectory datasets, we modify the event-based simulator GTMobiSim [70] to generate mobility traces on real road

networks. We use maps of the real road networks of North West Atlanta (ATL) and West San Jose (SJ) which are obtained from U.S. Geological Survey data [85]. ATL map is a rural/suburban road network with low density of network topology. SJ map is an urban road network with a higher density of network topology, which means higher density of road segments and junctions. Concretely, we use the random trip model. The start and end locations in each trip of a mobile object are randomly chosen from a predefined set of hot spots on the map. Each object moves independently of others, under the speed limit and speed distribution defined for the road segments. The speed limits are set for each type of road at 30mph for residential, 55mph for highway, 70mph for freeway and 30mph for freeway interchange. Object speeds are chosen from a Gaussian distribution with a standard deviation of 0.2 times the mean, which is the road segment speed limit. The objects are simulated to travel following the shortest routes to reach their destinations as in real life traveling. The trajectory locations of an object are recorded every 5 seconds during its trip. Thus, the trajectory datasets generated reflect user movement along the real world road networks.

Figure 22(a) and Figure 22(a) show the 1000 mobility traces generated on the road networks of ATL and SJ, called ATL1 and SJ1 datasets, plotted as light green polylines on ATL map and on SJ map respectively. We sample ATL1 and SJ1 datasets with duplicates to produce datasets of size from 1000 to 10000 trajectories for each road network in our experiments. ATL1 and SJ1 are also used as sample datasets to select the optimal dimensionality for the TrajMap trajectory transformation. The value of the best d to use in different implementations of TraceMob is reported in Table 6. The cell size is by default set to $[186m \times 220m]$ and $[176m \times 218m]$, which are equivalent to the same grid size of $[2^6 \times 2^6]$, for ATL and SJ maps respectively, so that the average number of road segments residing in a grid cell is in the range of $[1, 3]$. In Section 3.8.5 we will evaluate the effectiveness of TraceMob by varying grid cell sizes.



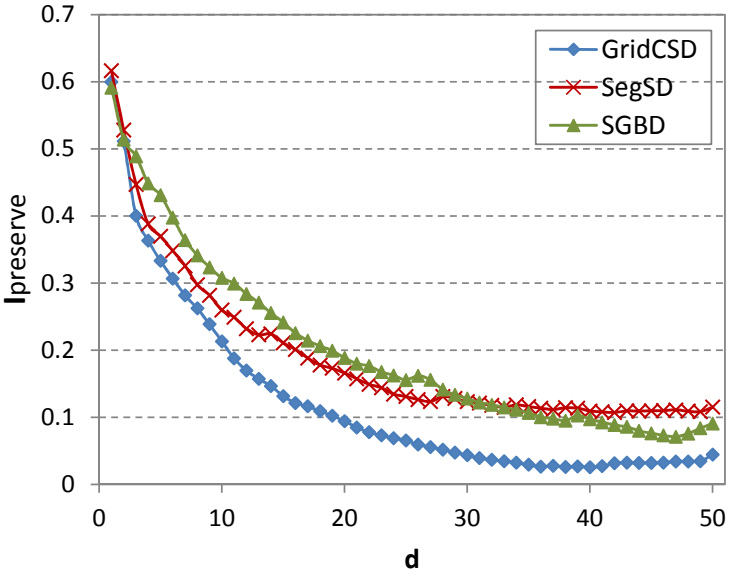
(a) ATL1 mobility traces



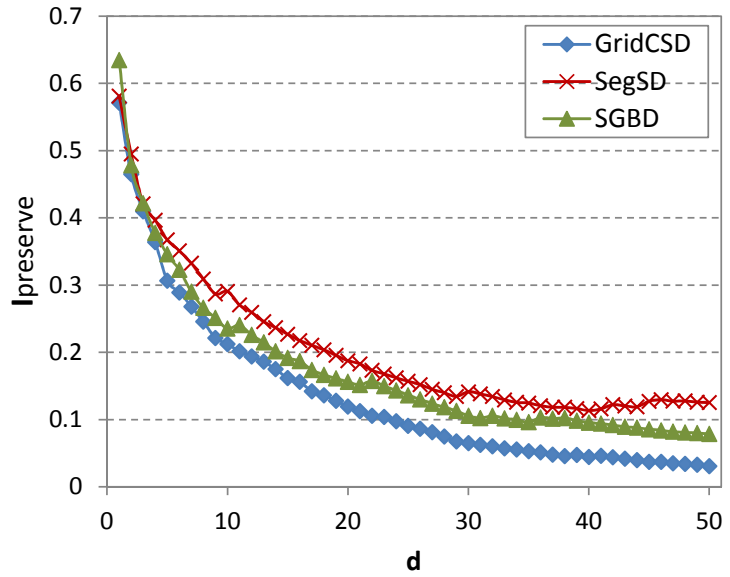
(b) SJ1 mobility traces

Figure 22: Sample synthetic datasets

3.8.2 Spatial Structure Preservation



(a) sample data of ATL1



(b) sample data of SJ1

Figure 23: Computing $I_{preserve}$ to choose d^*

Figure 23 shows the $I_{preserve}$ computed for each d from 1 to 50 to find the best d for all three TraceMob implementations, GridCSD-TraceMob, SGBD-TraceMob and SegSD-TraceMob. We list the optimal d for both ATL1 and SJ1 in Table 6. For example, in GridCSD-TraceMob which uses the GridCSD distance function performing

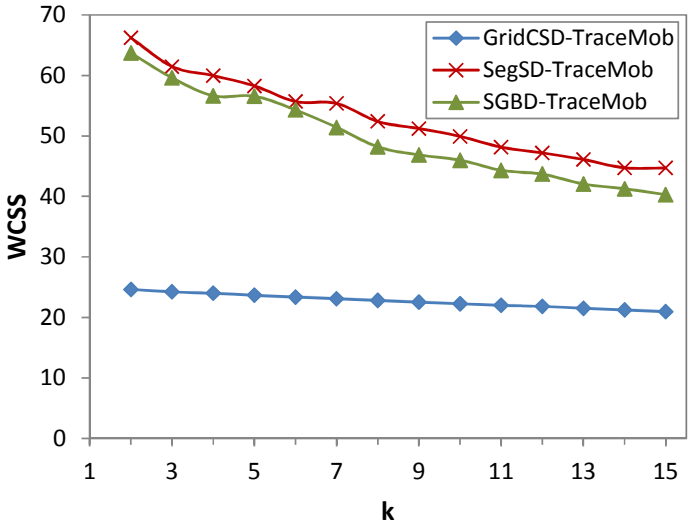
on Atlanta datasets, the best d is 40, which minimizes $I_{preserve}$. We can see from Figure 23 that as d increases, $I_{preserve}$ decreases, indicating that all the transformations preserve the spatial structure of the original trajectory better with increasing d until d reaches the optimal point in the given range. The values of $I_{preserve}$ for GridCSD decrease faster and are smaller than those of SegSD and SGBD, which shows that using GridCSD for the transformation helps better preserve the spatial structure of the trajectories than using SGBD and SegSD. The reason is that SegSD produces distances of 1 (recall that all three distance functions are normalized between 0 and 1) for all pairs of trajectories that do not contain overlapping road segments no matter how close they are in the road network space. SGBD performs slightly better than SegSD because it uses the grid-based representation and considers the common grid cells in two trajectories, which can correct the problem of two trajectories which are close but do not share common road segments. However, it still produces inaccuracy where trajectories do not have overlapping cells or their mergeable cells dominate. This is corrected by the GridCSD function, which considers both the common and mergeable grid cells, as well as adding weights to favor longer common cell sequences. Therefore, GridCSD describes accurately the spatial structure of the original trajectory dataset and also well preserves the spatial structure of data in the image space.

3.8.3 Cluster Validation

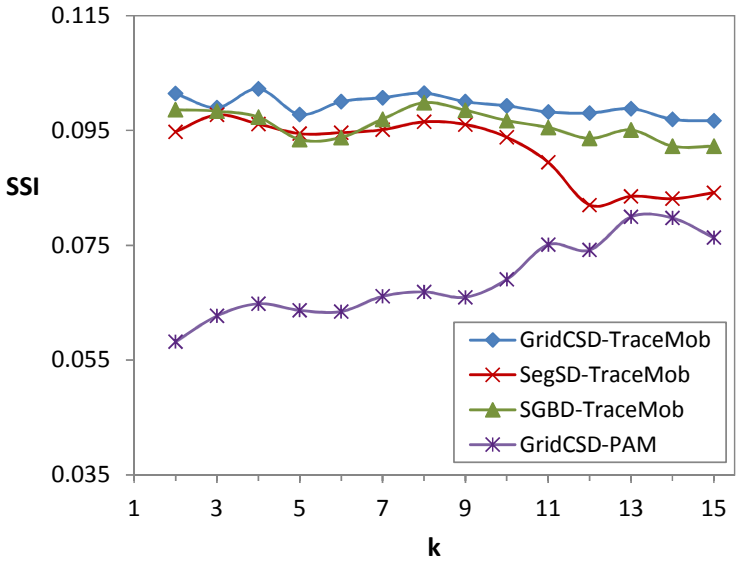
We show the evaluation of clustering results by analyzing both the WCSS (within cluster sum of square) and the SSI (spatial Silhouette index). The value of k supplied for k-means clustering varies from 1 to 15. Figure 24 displays the results for the ATL1 dataset. Recall that SSI is proportional to cluster quality, while WCSS is reversely proportional to cluster quality. Figure 25(a) shows that the clustering result produced by SGBD-TraceMob has slightly better quality than that of SegSD-TraceMob for all

values of k. While there is a big gap in all values of WCSS produced by GridCSD-TraceMob compared to the other two approaches. Thus, GridCSD-TraceMob produces a better clustering structure than SGBD-TraceMob and SegSD-TraceMob. This results from the capability of the GridCSD function to better describe and preserve the spatial structure of trajectory data, as demonstrated above.

In Figure 24(b), for most values of k, GridCSD-TraceMob produces results with the highest quality in terms of SSI (Spatial Silhouette Index). SGBD-TraceMob stays close to GridCSD-TraceMob for most of k values. However, segSD-TraceMob starts to show decreasing SSI values when k reaches 9. However, there is no big gap appearing in the SSI curves of the clustering results in the road network space. This interesting fact is also displayed in Figure 25 for the SJ1 dataset. It can be explained by the difference between trajectories and multidimensional points. While a point in a multidimensional space is literally isolated and does not interfere with other points, a trajectory occupies a specific amount of space in the road network that can overlap with many other trajectories. That is why the clustering structure in the road network space is less pronouced than in a multidimensional space. Also, GridCSD-PAM yields the lowest values of SSI in all cases showing that performing clustering directly in the road network space introduces inaccuracy in discovering trajectory clusters, in which it is hard to compute cluster centers accurately. Therefore, it is better to map trajectories to d-dimensional points in an Euclidean space before performing clustering. GridCSD-TraceMob remains to be the highest quality in terms of SSI for all k values, whereas GridCSD-PAM shows the lowest SSI values, demonstrating the effectiveness of TraceMob trajectory clustering framework compared to directly clustering whole trajectories using GridCSD-PAM.



(a) WCSS

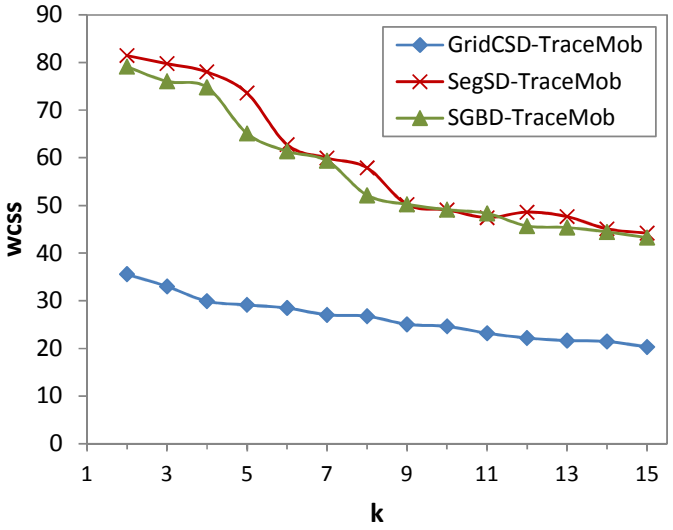


(b) SSI

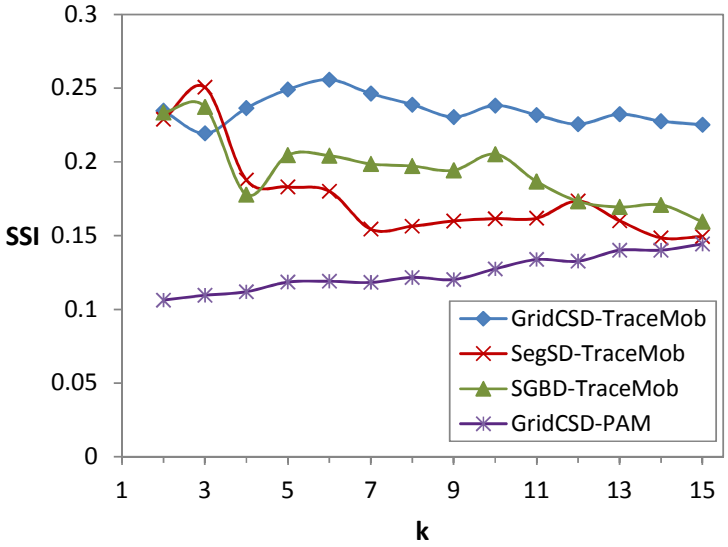
Figure 24: Cluster validation for ATL1 dataset

3.8.4 Performance Evaluation

This set of experiments is to evaluate the efficiency of GridCSD-TraceMob in terms of time complexity. For a fair comparison, we use the same values of $d = 50$ to run GridCSD-TraceMob, SegSD-TraceMob, SGBD-TraceMob instead of using the best d used in the previous experiments. We also use the same value of $k = 10$. We compare the running time of the three implementations on ten datasets of varying sizes from



(a) WCSS



(b) SSI

Figure 25: Cluster validation for SJ1 dataset

1000 to 10000 trajectories. Figure 26 shows that GridCSD-TraceMob and SGBD-TraceMob are scalable and run up to two orders of magnitude faster than SegSD-TraceMob and GridCSD-PAM as the data size increases for both ATL1 (suburban) and SJ1 (urban). This shows the inefficiency of perform clustering directly in the road network space where the centroid of a set of trajectories is costly to compute, given that partition based clustering requires iterative computation until it converges.

Figure 27 displays the statistics about the number of cells in each grid-based representation and the number of road segments in each segment-based representation of a trajectory in ATL and SJ datasets. It shows that the road segment cardinality of a trajectory approximately doubles its cell cardinality. Also, the road segment cardinality of a trajectory approximately doubles its cell cardinality since one grid cell covers a number of segments in the road network. Moreover, SegSD needs to access the geometric points of a road segment in the map to compute its length, which ranges from 2 to 34 points in ATL map, and 2 to 26 points in SJ maps. Thus, each GridCSD or SGBD computation requires much less time than SegSD computation (recall the complexity analysis in Section 3.4). Since the performance of the transformation phase heavily depends on the input distance function, that makes GridCSD-TraceMob and SGBD-TraceMob achieve big savings in time compared to SegSD-TraceMob.

Figure 28(a) compares the average running time in seconds of k-means clustering by varying the sizes of trajectory data for all three distance based schemes. The measurement reported for each scheme is the average running time for the trajectory image datasets of both ATL and SJ maps with $d = 50$. It shows that the time complexity of the k-means clustering is increasing as the size of trajectory dataset increases for all three approaches but for each given dataset, the running time for k-means clustering is quite similar for all three distance metrics. Furthermore, comparing with the running time of the transformation phase in Figure 26, we see that the running time of k-means clustering in all three cases is negligible (ranging from 0.5s to 14s) compared to that of *TrajMap* mapping (SGBD-TraceMob takes from 64s to 3918s for mapping). Thus, the transformation phase mainly contributes to the cost to run each TraceMob scheme, which combines the running time of transformation phase and clustering phase. Figure 28(b) plots the total running time of three TraceMob schemes along with that of the GridCSD-PAM scheme on ATL datasets.

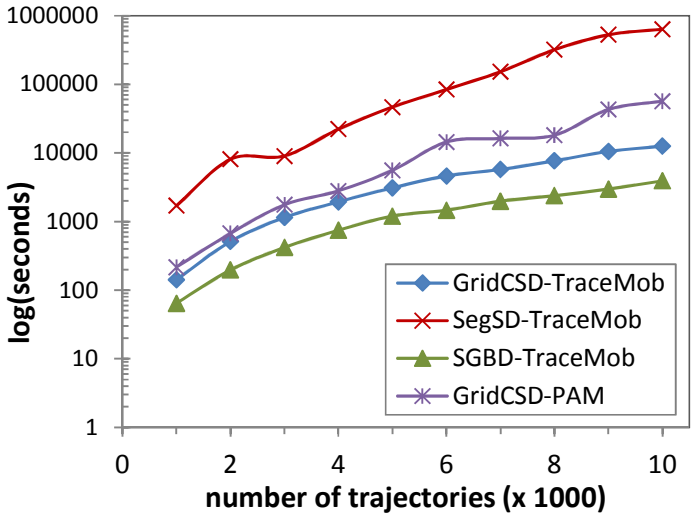
We can see that the relative performances of four schemes are similar to the results shown in Figure 26. Combining Figure 26 with Figure 25, we show that the grid-based trajectory clustering approach consumes less trajectory distance computations while yielding better clustering quality.

This results from the linearity of k-means clustering in an Euclidean space, where only Euclidean distance computation is required and its ability to quickly converge when we use careful seeding, e.g., we use k-means++ algorithm [14] to select initial centroids in our implementation. Therefore, in the case when there are several desirable values of k, we can quickly obtain the clustering results since we only need to perform k-means clustering on the set of multidimensional points instead of repeatedly performing the clustering for different k directly on the complex trajectory data.

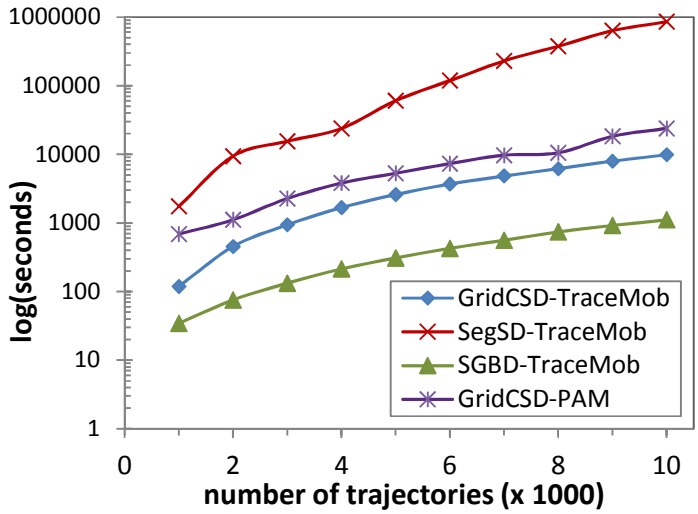
3.8.5 Varying cell sizes

Finally, we measure the performance of GridCSD-TraceMob by varying the settings of grid cell size for both suburban map (ATL) and urban map (SJ). We run GridCSD-TraceMob over the dataset of 5000 trajectories for each map. Figure 29(a) shows the measurement results for different settings of grid size. At the grid size of 32×32 (i.e., $2^5 \times 2^5$), each cell covers an average of 8 road segments in ATL map and 13 road segments in SJ maps. At the grid size of 512×512 (i.e., $2^8 \times 2^8$), each cell covers an average from 0 to 1 road segments for both maps. The results show that the running time increases as the size of the grid cell decrease, which means that the number of cells in the grid increases. The increased running time can be attributed to the increased number of cells representing each trajectory as the cell size decreases, and thus trajectory distance computation takes longer time during *TrajMap* transformation.

We can also see that TraceMob runs faster on ATL map, which is a suburban



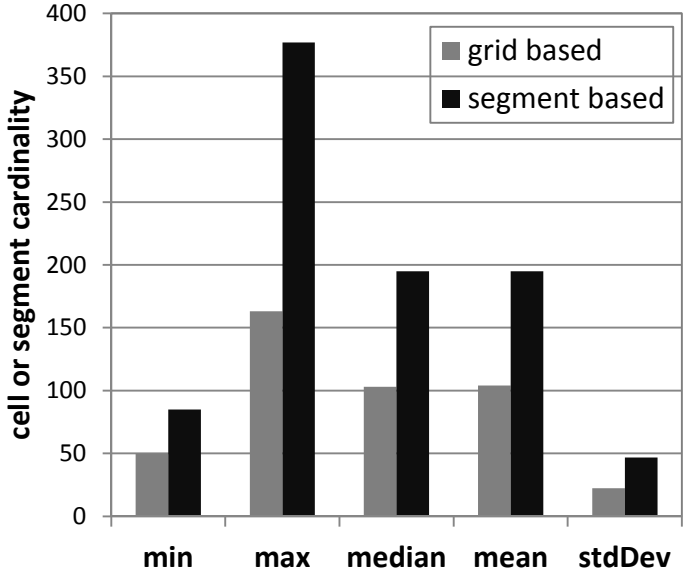
(a) Atlanta datasets.



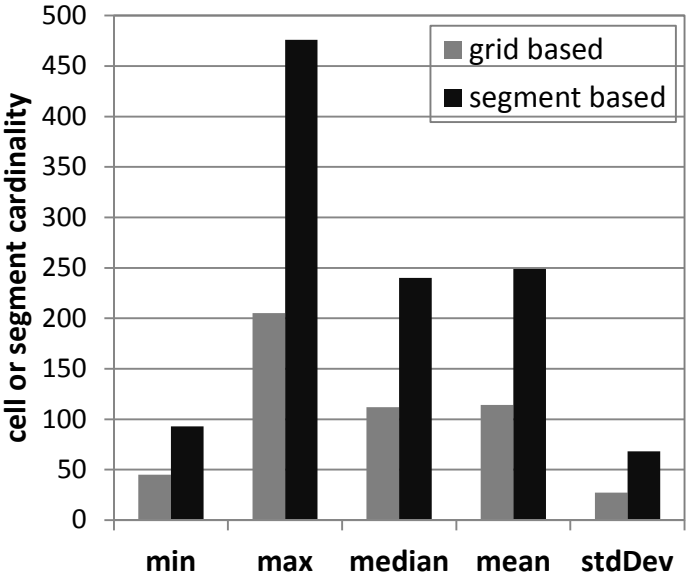
(b) San Jose datasets.

Figure 26: Running time of the transformation phase with Y-axis logarithmic.

road network with low density of network topology, than on SJ map, which is an urban road network with high density of network topology. In addition, the running time increases less for ATL map than for SJ map when the grid cell size decreases. Recall the statistics reported in Figure 27, we can see the length in terms of the number of cells per trajectory on average in ATL datasets is shorter than that in SJ datasets. Also, the number of road segments intersects with a grid cell of the same size on ATL map is smaller than that on SJ map. Thus, GridCSD computation is



(a) Atlanta datasets.

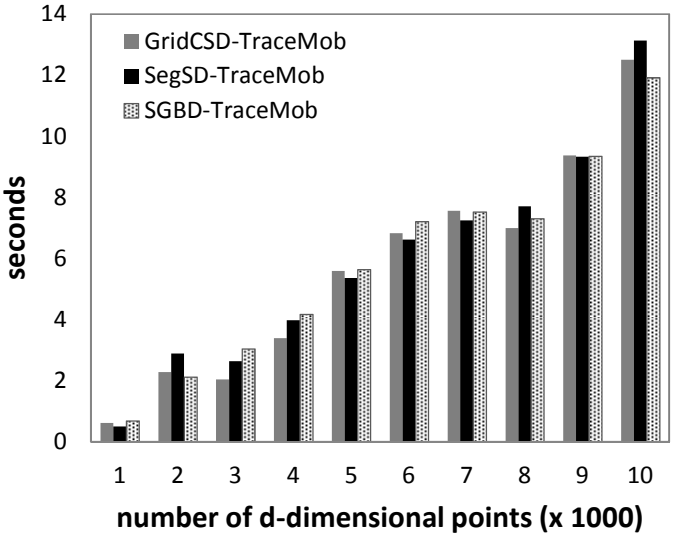


(b) San Jose datasets.

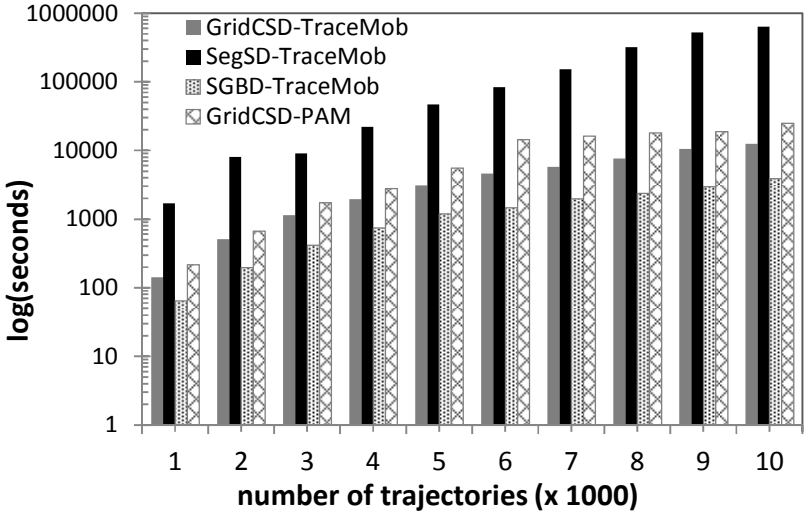
Figure 27: Average number of cells/segments per trajectory

faster for trajectories in ATL map than in SJ map, which is further confirmed by the results shown in Figure 29.

Figure 29(b) and Figure 29(c) show the clustering quality measurement in terms of WCSS and SSI respectively by varying cell sizes. We can see that choosing too large cell size or too small cell size will result in poor clustering quality. The most



(a)



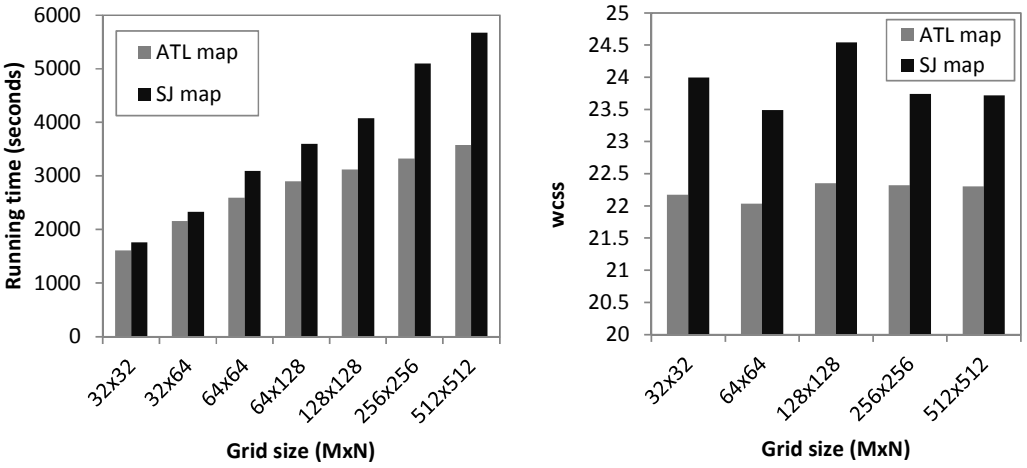
(b)

Figure 28: (a) Average running time of k-means clustering phase, (b) Total running time with Y-axis logarithmic. (Atlanta datasets)

suitable grid size ($2^6 \times 2^6$) for these experiments gives better clustering structure for the trajectory dataset with the lowest value of WCSS and the highest value of SSI.

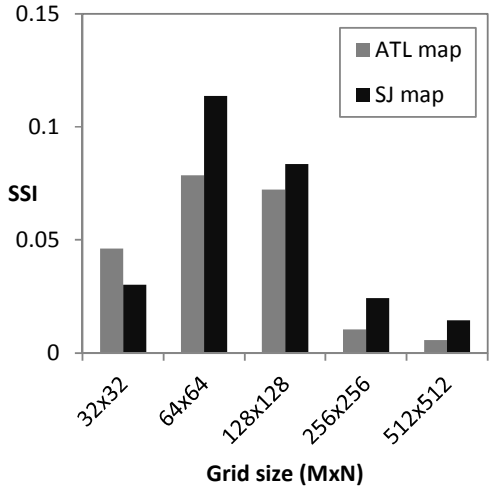
3.9 Conclusion

We have presented TraceMob, a methodical approach to clustering whole trajectories of mobile objects traveling in road networks. This chapter makes three original



(a) Running time

(b) WCSS



(c) SSI

Figure 29: Varying size of grid cells

contributions. First, we introduce GridCSD to measure the distance between two trajectories of varying lengths. It fully captures the spatial proximity of trajectories in a road network and works effectively for both trajectories that share common road segments and trajectories which are non-overlapping but still close to each other. Second, we develop *TrajMap* by extending FastMap, which transforms mobile object trajectories in a road network space into d-dimensional points in an Euclidean space while preserving the original pairwise distances of road network trajectories.

We achieve this objective by carefully selecting the initial pivot trajectories and iteratively refining the dimensionality d during the transformation. Third but not the least, we evaluate the clustering quality in both projected trajectory image space and the road network trajectory space. Extensive experiments demonstrate the utility of our three phase framework for whole trajectory clustering and show that GridCSD-TraceMob outperforms the simple grid based approach (SGBD-TraceMob), segment based approach (SegSD-TraceMob) and direct clustering over road network trajectories (GridCSD-PAM) in terms of both clustering quality and time complexity.

CHAPTER IV

TRAJPOD: FAST LOCALITY-AWARE TRAJECTORY PATTERN MINING

In recent years, trajectory data of mobile objects in road networks (MO trajectories) has become ubiquitous, which presents exciting mining challenges for studying the dynamics of human mobility. In this chapter, we investigate the problem of finding the spatial-temporal correlation in the movement of mobile objects through mining trajectory sequential patterns. Since MO trajectories are temporal sequences of location points of mobile objects moving in constrained road networks, there is a high degree of overlap in their temporal orders and spatial proximity, in addition to the large number of location points per trajectory. Therefore, applying existing sequential pattern mining algorithms for MO trajectories can suffer from exponential growths in space and computation. We define trajectory patterns as time-ordered sequences of semantic spatial units and propose a novel algorithm, called TrajPod, to extract the complete set of frequent trajectory patterns from MO trajectory data. To handle the complex spatial-temporal characteristics of MO trajectories, TrajPod uses a vertical format of the trajectory database, i.e., trajectory id-lists, which are partitioned into locality-aware sublists. In addition, TrajPod uses a space-efficient representation for multiple occurrences of a semantic spatial unit in a trajectory with respect to each sublist. These techniques allows TrajPod to perform trajectory pattern mining efficiently with early candidate pruning and fast support counting. Extensive experiments with varying size datasets of mobility traces generated from real road networks demonstrate that TrajPod is efficient and outperforms existing sequential pattern mining approaches up to an order of magnitude.

4.1 Background and Introduction

We first give an overview on the background of the traditional sequential pattern mining research. We then address the problem of mining mobile object trajectory patterns and present our approach to the problem.

4.1.1 Sequential Pattern Mining

Sequential pattern mining has long been recognized as a primary mining task to analyze sequential data including purchase history, web logs and biological data. The problem of sequential pattern mining has been studied extensively since it was first introduced by R. Agrawal and R. Srikant [10] in 1995 for *shopping basket* data. The input data is a set of data sequences. Each data sequence Seq is a time-ordered list of events where each event consists of a set of items and identified by a sequence identifier $SeqId$, formally represented by $Seq = \{SeqId, OrderedList < Event >\}$. An example of a sequence in the shopping basket data is $\{Sid, < (cheese, diaper) \rightarrow (milk, bread) \rightarrow (beer, coke) \rightarrow (salt) >\}$, in which an event is a transaction consisting of grocery items. A result from sequential pattern mining can be "70% of the customers buy diapers and cheese, then also buy beer", which means 70% of the input sequences contains the subsequence $< (cheese, diaper) \rightarrow (beer) >$. The subsequence $< (cheese, diaper) \rightarrow (beer) >$ is called a *frequent sequence* given a support threshold $\alpha < 70\%$. Such discovered knowledge presents the associations between items in the sequence data, which is extremely helpful for applications in recommendation systems, business prediction and planning, store reorganization and marketing strategy.

The downward-closure property, Apriori, that an itemset is frequent only if all of its subsets are frequent, which was observed in [9], has been the basis for existing sequential pattern mining algorithms. Approaches in sequential pattern mining mainly fall into two categories, namely, Apriori-based and projection-based pattern growth.

Apriori-based algorithms [10,15,80,97] mainly exploit Apriori property to compute frequent patterns in a generate-and-test fashion, in which all frequent k-patterns are mined from the set of frequent $(k-1)$-patterns starting from frequent 1-patterns. Early Apriori-based algorithms, e.g., AprioriAll [10], GSP [80], make multiple passes through the database to count the support of the current candidates, which yield undesirable performance due to expensive I/O cost. SPADE [97] and SPAM [15] employ a vertical format of the database, which is the occurrence lists of the items in the database, and compute the supports based on the corresponding operators over these lists, reducing the number of database scans to one or two. SPADE has to perform whole id-list joins to test large sets of candidate patterns. SPAM speedups support counting by using bitwise operators. However, its bitmap data structure is very space-inefficient because every candidate is represented by bit vectors whose size is the same as the number of the data sequences.

Projection-based algorithms, such as FreeSpan [45] and PrefixSpan [69], recursively project sequence databases into smaller projected databases based on the current frequent patterns and examine frequent patterns locally in the projected databases to grow patterns. This pattern growth process also implies the Apriori principle that any sequence whose projected itemset is a super-pattern of a trivial itemset is also a trivial pattern. Although FreeSpan searches smaller projected databases than the original database, it still has to keep the whole sequence and check the growth of a pattern at any split point in a candidate pattern, which is costly. PrefixSpan improves over FreeSpan by checking only the prefixes of patterns and projects their corresponding suffixes into prefix-projected databases where it can explore the frequent patterns locally. The projected databases in PrefixSpan shrink faster than those of FreeSpan. Although there is no candidate patterns generated, Prefix has to generate and explore the entire projected databases recursively which can consume a lot of space and processing time.

SPADE, SPAM as the Apriori-based representatives and PrefixSpan as the projection-based representative are the most efficient algorithm benchmarks to date.

4.1.2 Mobile Object Trajectory Pattern Mining

In recent years, trajectory data of mobile objects in road networks (MO trajectories) has become ubiquitous due to advances in positioning technologies, the dramatic growth in the number of active mobile devices and advances in the field of location based services and applications. The huge source of MO trajectory data presents exciting mining challenges in order to study the dynamics of human mobility. In this chapter, we investigate the problem of finding the spatial-temporal correlation in the movement of mobile objects through mining trajectory sequential patterns. Our goal is to find the complete set of trajectory patterns, which helps discover interesting association rules between locations in the given trajectory dataset. The discovered trajectory sequential patterns can be used in broad applications such as trip recommendation, location prediction, location-based advertisement.

Intuitively, existing sequential pattern mining algorithms, represented by SPADE [97], SPAM [15] or PrefixSpan [69], can be applied to mine trajectory patterns since MO trajectories are temporal sequences of location points representing moving paths of mobile objects in road networks. Here, each event in a trajectory has only one item, which is a road network location, as a user can only be at one location at a time, instead of multiple items per event as in traditional sequence data. Moreover, since MO trajectories are constrained by the road network as mobile objects can only move within road segments, there is a high degree of overlap in their temporal orders and spatial proximity. Therefore, applying general sequential pattern mining algorithms to MO trajectories, without considering and utilizing their spatial-temporal characteristics, can suffer from exponential growths in space and computation. We carefully study the spatial-temporal characteristics of trajectory data to define the

following key design guidelines for an efficient trajectory pattern mining algorithm.

First of all, we define a trajectory pattern as a sequence of semantic spatial units. Due to the high location tracking rate, the number of location points recorded per trajectory can be very large. Consecutive points in a trajectory can be dense and belong to the same semantic spatial unit on a road segment, e.g., a region passing a shopping mall. Hence, we aim to mine a set of meaningful trajectory patterns as sequences of semantic locations instead of raw GPS locations. We define trajectory pattern as a time-ordered sequence of semantic spatial units, given a set of spatial semantic units associated with a road network. Each semantic spatial unit can be a road segment or a grid cell in a grid overlay of the road network. It covers a portion of the road network locations in a given trajectory dataset. An example of a MO trajectory pattern which we aim to mine can be "70% of the mobile objects go to A, then to B, then also go to C during their trips" where A, B, C are the semantic spatial units in the road network.

Second, we use a vertical layout of the trajectory dataset for space efficiency. Beside the aforementioned property of one-item event, we observe that mobile objects usually travel following shortest paths. Thus, there can be a lot of overlapping parts among MO trajectories within the same road network. A location can have a large number of occurrences across different trajectories. This makes the vertical layout of a trajectory dataset, i.e., the occurrence id-list associated with the semantic spatial units, a space-efficient way to manipulate trajectory data.

Last but not least, we take a locality-aware approach in support counting for computational efficiency. We observe that many MO trajectories within a road network can share the same ending semantic locations, for instance, trajectories of people living in the same neighborhood or working in the same company. Thus, we organize each occurrence id-list into locality-aware sublists. In addition, we use a space-efficient representation for multiple occurrences of a semantic spatial unit in a trajectory with

respect to each sublist. These techniques help reduce the algorithm's search space and greatly improve the performance by incorporating early candidate pattern pruning and efficient support counting.

Based on these design guidelines, we construct TrajPod, an algorithm for TRAJectory pattern discovery using locality-aware Partitioning of the Occurrence iD-lists. TrajPod efficiently discovers the complete set of trajectory patterns by utilizing the spatial characteristics of MO trajectories for early and deep pruning candidate patterns along with depth-first traversal of its search space. While SPADE scans the whole occurrence id-lists and PrefixScan scans the entire projected data recursively for each frequent prefix, TrajPod only performs on the partitioned occurrence id-lists which greatly reduces the search space and support counting computation cost. A comprehensive experimental evaluation demonstrates the space and computational efficiency of TrajPod as it has better memory utilization and runs up to an order of magnitude faster than state-of-the-art approaches in traditional sequential pattern mining.

The rest of this chapter is organized as follows. Section 4.2 presents the reference road network model and MO trajectories in the form of sequences of semantic spatial units. The problem statement and basic concepts of mining MO trajectory pattern are introduced in Section 4.3. Section 4.4 gives an in-depth presentation of our approach to the trajectory mining problem. We report our experimental results in section 4.5, discuss related work in Section 4.6 and conclude the chapter in section 4.7.

4.2 Road network and MO trajectories

In this section, we present the basic model of a road network and the definition of MO trajectories. We then describe the use of a *semantic spatial coverage* to represent a MO trajectory as a sequence of semantic spatial units for our pattern mining purpose.

4.2.1 Road Network Model

A road network is modeled by a single directed graph $G = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V} = \{v_0, v_1, \ldots, v_N\}$ is a set of road intersections and $\mathcal{E} = \{(sid, v_i v_j) | v_i, v_j \in \mathcal{V}\}$ is a set of directed edges connecting the road intersections.

Each edge $e = (sid, v_i v_j) \in \mathcal{E}$ is identified by the road segment id sid which connects two road intersections v_i and v_j in the real road network.

We define a *road network location* as a tuple of three elements $l = (sid, (x, y), t)$, where sid is the identifier of road segment where the object resides, (x, y) is the geometric coordinates of the object's location, and t is the timestamp when the location is recorded.

When a mobile object, e.g., a person equipped with a GPS/Wifi-enabled mobile device (such as a car or a smartphone), moves in a road network G, the locations recorded during its trip form a trajectory and formally defined as follows.

Definition 20 *(Road Network Trajectory)* A road network trajectory Tr, denoted by $Tr = (trid, l_1 l_2 ... l_L)$, is a time-ordered sequence of L road network locations l_i $(1 \leq i \leq L)$ and is uniquely identified by a trajectory identifier $trid$.

4.2.2 Semantic Spatial Coverage

We define a *semantic spatial coverage* associated with a road network G, denoted as $SSC(G)$, as a set of *semantic spatial units* which covers all the road network locations in G. Each semantic spatial unit, called s-unit, is a spatial area representing a semantic network proximity where a group of road network locations reside. By distance measure, two road network locations within the same s-unit are closer to each other than the distance between two locations in two different s-units. For instance, a grid cell, a road segment, a hotspot or point of interest in the road network can be considered a semantic spatial unit.

We argue that trajectory pattern mining should consider frequent sequences of semantic spatial units as more meaningful than frequent sequences of raw road network locations. Thus, transforming MO trajectories into sequences of s-units is a necessary preprocessing step for trajectory pattern mining.

In general, given an $SSC(G)$, a MO trajectory can be formatted as a time-ordered list $Tr = (trid, u_1u_2...u_n)$ in which each u_i is an s-unit in $SSC(G)$. We present in the followings two SSC schemes to represent MO trajectories as sequences of s-units: segment-based and grid-based.

4.2.2.1 MO trajectories as sequences of road segments

Trajectories collected via GPS sensing are often quite long in terms of the number of the recorded locations, especially with frequent periodic sensing. Long trajectories are those consisting of a large number of location points. By utilizing road network characteristics, we can abstract road network trajectories using road segments without loss of data quality. Since a mobile object can only move along consecutive road segments, a trajectory $Tr = (trid, l_1l_2...l_L)$ in a road network $G = (\mathcal{V}, \mathcal{E})$ can be alternatively represented by its road segment sequence, denoted by $Tr = (trid, e_1e_2...e_R)$ $(e_i \in \mathcal{E}, 1 \leq i \leq R, R \leq min(L, |\mathcal{E}|))$.

4.2.2.2 MO trajectories as sequences of grid cells

Given a road network G, we refer to the minimum bounding rectangle (MBR) region that covers the entire road network as the *universe of discourse*, which is defined by $U(G) = Rect(X, Y, W, H)$, where X is the x-coordinate and Y is the y-coordinate of the lower left corner of the MBR corresponding to the universe of discourse. W is the width and H is the height of the universe of discourse. X, Y, W and H are system level parameters to be set at the system initialization time.

Grid Overlay: We partition $U(G)$ into a grid of contiguous cells to form a grid topology of the road network G. We consider a uniform grid topology with identical

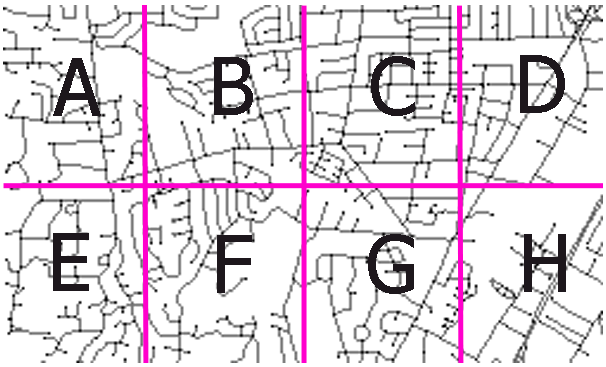


Figure 30: A grid overlay of a sample road network G_s as its SSC

rectangular cells of size $\alpha \times \beta$. Formally, a grid corresponding to the universe of discourse $U(G)$ can be defined as $\mathcal{A}_{Grid(G)}(U(G), \alpha, \beta) = \{A_{i,j} : 1 \leq i \leq M, 1 \leq j \leq N,\ A_{i,j} = Rect(X + i * \alpha, Y + j * \beta, \alpha, \beta), M = \lceil H/\alpha \rceil, N = \lceil W/\beta \rceil\}$. α and β are system parameters that define the cell size of the grid. $A_{i,j}$ is an $\alpha \times \beta$ rectangular area representing the grid cell that is located on the ith row and jth column of the grid $\mathcal{A}_{Grid(G)}$. Each cell entry $A_{i,j}$ is uniquely identified by a cell identifier cid, which is an integer in $[1, M * N]$.

$$A_{i,j}.cid = (i-1) * N + j, i \in [1, M], j \in [1, N] \tag{23}$$

Position to Grid Cell Mapping: Let $l.pos$ denote the geometric coordinates (x, y) of a road network location $l = (sid, (x, y), t)$. Let $A_{i,j}$ denote a cell in the grid $\mathcal{A}_{Grid(G)}$. $Pmap(l.pos)$ is a road network location to grid cell mapping, defined as $Pmap(l.pos) = A_{\lceil \frac{pos.x - X}{\alpha} \rceil, \lceil \frac{pos.y - Y}{\beta} \rceil}$.

Each trajectory and its road network locations are indexed by the grid matrix $\mathcal{A}_{Grid(G)}$. Thus, a trajectory $Tr = (trid, l_1 l_2 ... l_L)$ can be represented by a time ordered sequence of grid cells covering its locations, denoted by $Tr = (trid, c_1 c_2 ... c_n)$, where each $c_k (1 \leq k \leq n)$ is a grid cell covering a subsequence of $l_1 l_2 ... l_L$, denoted by $l_{i_1} \ldots l_{i_m}$. For example, given a uniform grid of size 2×4 covering a sample road network G_s shown in Figure 30, we have 8 grid cells A, B, C, D, E, F, G, H as 8 s-units. The trajectory of a mobile object that moves consecutively through A, B, F, G

and H will be represented by the sequence $ABFGH$ covering the trajectory path.

For presentation convenience, all the definitions in the rest of the chapter are assumed to be given in the context of a road network $G = (\mathcal{V}, \mathcal{E})$ and its semantic spatial coverage $SSC(G)$. Without loss of generality, we illustrative our approach using examples given a sample grid-based SSC. We also use the alphabetical symbols to denote the identifiers of s-units when no confusion occurs.

4.3 Problem Description

In this section, we begin with defining the necessary terminology and scope of the trajectory pattern mining problem we are solving.

4.3.1 Trajectory Pattern Mining

Given a MO trajectory data set $\mathcal{T} = \{TR_1, TR_2, \ldots, TR_N\}$ consisting of N trajectories in a road network $G = (\mathcal{V}, \mathcal{E})$, along with its semantic spatial coverage $SSC(G)$. We will use the s-unit based representation for the given MO trajectories where each trajectory $TR_i \in \mathcal{T}$ is a sequence of s-units from $SSC(G)$, denoted by $Tr_i = (trid, u_1u_2...u_n)$.

We use $traj(trid)$ and $trid$ interchangeably to refer to the trajectory with identifier $trid$ in the trajectory dataset. Throughout the chapter, we use the symbol $||$ to denote the unit cardinality of an object. For instance, $|Tr_i|$ is used to denote the s-unit cardinality of trajectory Tr_i.

Definition 21 *(Trajectory Pattern)* A *trajectory pattern* α is an ordered list of s-units, denoted by $u_1u_2 \ldots u_k$, where u_i $(1 \leq i \leq k)$ is an s-unit in $SSC(G)$.

For $u_i \in \alpha$ and $u_j \in \alpha$, we denote $u_i < u_j$ if u_i occurs before u_j in pattern α. The number of s-units in a pattern is the length of the sequence. A pattern of length k is called k-pattern. By definition, a trajectory is also a trajectory pattern itself.

A pattern β is said to contain pattern α and is called a *super-pattern* of α, denoted as $\alpha \preceq \beta$, if $u_i \in \alpha$ then $u_i \in \beta$ and the order of s-units in α is preserved in β, i.e., if $u_i < u_j$ in α then we also have $u_i < u_j$ in β. α is said to occur in β and is called a *sub-pattern* of β. For example, $u_1u_2u_3u_4$ is a supper-pattern of u_2u_4.

Definition 22 *(Pattern Occurrence)* An occurrence of a trajectory pattern α with respect to a trajectory T_i is presented by a pair $(trid, pid)$ where $trid$ is the identifier of T_i and pid is where α ends in T_i, i.e., the position of the last s-unit of pattern α in the ordered s-unit list of trajectory T_i.

We use $fpos(\alpha, T_i)$ to denote the function that returns all the pid values in the occurrences of pattern α in trajectory T_i. The occurrence cardinality $|fpos(\alpha, T_i)|$ is 0 if T_i does not contain α, otherwise $|fpos(u_i, T_i)| \geq 1$. For example, given trajectory $T_i = BABCBE$, we have $fpos(AB, T_i) = \{3, 5\}$ and $|fpos(AB, T_i)| = 2$ since pattern AB occurs twice in T_i including one ends at the third position ($B\mathbf{AB}CBE$) and another one ($B\mathbf{A}BC\mathbf{B}E$) end at the fifth position in the s-unit sequence $BABCBE$.

Given a k-pattern $\alpha = u_1u_2 \dots u_{k-1}u_k$, the prefix of α, denoted by $pref(\alpha)$, is the $(k-1)$-subpattern consisting of the first $k-1$ s-units in α: $pref(\alpha) = u_1u_2 \dots u_{k-1}$. Let $last(\alpha)$ denote the function extracted the last s-unit in α, i.e., $last(\alpha) = u_k$.

For each pair of $(k-1)$-patterns having the same prefix α_1 and α_2, let say $\alpha_1 = u_1u_2..u_{k-2}a$ and $\alpha_2 = u_1u_2..u_{k-2}b$, let $\alpha_1 \rightarrow \alpha_2$ denote the *merge operator* that generates the k-pattern $u_1u_2..u_{k-2}ab$, i.e., $\alpha_1 \rightarrow \alpha_2 = u_1u_2..u_{k-2}ab$. We call α_1 and α_2 the *base patterns* of α where α_1 is the *left base* and α_2 is the *right base* corresponding to their positions in the *merge operator*.

We call the number of trajectories in $\mathcal{T}$ which contain α the absolute support of α, denoted by $abs\_sup(\alpha)$. The *support* of a trajectory pattern α is the fraction of trajectories in $\mathcal{T}$ which contains α, denoted as $sup(\alpha)$ and is computed as:

$$sup(\alpha) = abs\_sup(\alpha)/|\mathcal{T}| \tag{24}$$

Table 7: Example trajectory dataset $\mathcal{T}_s$

Trajectory ID	Grid Cell Sequence
T_1	ABF
T_2	$AFGH$
T_3	$EFGCH$
T_4	$ABEF$
T_5	EBF
T_6	CD
T_7	D
T_8	$ABFCBGH$

Given a minimum support *min_sup*, the absolute minimum support is denoted as *abs_minsup* and is computed as: $abs\_minsup = \lceil min\_sup \times |\mathcal{T}| \rceil$.

Problem Statement. The problem of trajectory pattern mining is that given a MO trajectory data set $\mathcal{T}$ and a minimum support *min_sup*, the goal is to extract the complete set of trajectory patterns from $\mathcal{T}$ that have support above *min_sup*. These trajectory patterns are called *frequent* patterns. All other patterns, which are not frequent, are said to be *trivial*.

Example 5 Consider the same grid structure in Figure 30. Let 8 grid-based trajectories as listed in Table 7 be the trajectory database $\mathcal{T}_s$ used as a running example of this chapter. Let the minimum support *min_sup* be 0.25, which means that a frequent pattern α must occur in at least 2 trajectories, i.e., $abs\_sup(\alpha) \geq 2$. Then a set of frequent patterns are generated as listed along with their absolute supports (in brackets) in Table 8 including 1-patterns, 2-patterns, 3-patterns and 4-patterns.

4.3.2 The Generate-and-Test Framework

We recall the popular generate-and-test framework for sequential pattern mining with SPADE as its best representative. We will give a brief summary of the framework and SPADE *in the context of trajectory pattern mining*, in which, sequences are MO trajectories in road network G and items are the s-units associated with road network G. As described by its name, this framework finds frequent patterns in two basic steps:

Table 8: Frequent patterns from $\mathcal{T}_s$

1-pattern	2-pattern	3-pattern	4-pattern
$A(4)$	$AB(3)$	$ABF(3)$	$AFGH(2)$
$B(4)$	$AF(4)$	$AFG(2)$	
$C(3)$	$AG(2)$	$AFH(2)$	
$D(2)$	$AH(2)$	$AGH(2)$	
$E(3)$	$BF(4)$	$FCH(2)$	
$F(6)$	$CH(2)$	$FGH(3)$	
$G(4)$	$EF(3)$		
$H(3)$	$FC(2)$		
	$FG(3)$		
	$FH(3)$		
	$GH(3)$		

candidate pattern generation and support counting for testing the candidates against the input data. Initially, 1-pattern candidates include all the s-units from the SSC of the road network. The database is scanned once to compute the support for these 1-patterns to output the set of frequent 1-patterns (with support above the given min_sup). Let denote the set of frequent k-patterns as $\mathcal{S}_k$. In general, $\mathcal{S}_k$ is formed by: (1) Joining $\mathcal{S}_{k-1}$ with itself to generate all k-pattern candidates, (2) Support for the k-pattern candidates is computed by making a complete scan of the database. This process is repeated until no more frequent patterns can be found. All existing algorithms use the Apriori principle as a popular pruning technique to reduce the combinatorial search space in step 1. According to Apriori principle, all sub-patterns of a frequent pattern are also frequent. Thus, if a k-pattern is trivial, there is no need to extend it further, which applies in step 1 as we only need to build $\mathcal{S}_k$ from the frequent patterns in $\mathcal{S}_{k-1}$. Also, a candidate k-pattern is pruned out before support counting if there exists one $(k-1)$-subpattern of it which is not in $\mathcal{S}_{k-1}$.

However, scanning the database multiple times for support counting is too costly for large datasets. SPADE improves step 2 of the framework by performing simple joins over the vertical id-lists, called id-lists for short, of the database for support counting of the candidates instead of scanning the dataset in each iteration.

Table 9: Vertical id-lists of the dataset $\mathcal{T}_s$

A	B	C	D	E	F	G	H
$T_1, 1$	$T_1, 2$	$T_3, 4$	$T_6, 2$	$T_3, 1$	$T_1, 3$	$T_2, 3$	$T_2, 4$
$T_2, 1$	$T_4, 2$	$T_6, 1$	$T_7, 1$	$T_4, 3$	$T_2, 2$	$T_3, 3$	$T_3, 5$
$T_4, 1$	$T_5, 2$	$T_8, 4$		$T_5, 1$	$T_3, 2$	$T_8, 6$	$T_8, 7$
$T_8, 1$	$T_8, 2$				$T_4, 4$		
	$T_8, 5$				$T_5, 3$		
					$T_8, 3$		

An id-list of a pattern α, denoted by $\mathcal{L}(\alpha)$, is a list of occurrences of pattern α in the given trajectories and is formally defined as $\mathcal{L}(\alpha) = \{(trid, pid) | \alpha \preceq traj(trid), pid \in fpos(\alpha, trid)\}$.

There can be multiple occurrences of a pattern α in a trajectory, resulting in multiple $(trid, pid)$ pairs with the same $trid$ corresponding to that trajectory.

The id-list of a s-unit u_i, denoted by $\mathcal{L}(u_i)$, is essentially the id-list of the 1-pattern consisting of only u_i.

Let $|\mathcal{L}(\alpha)|$ be the number of distinct $trid$ in the id-list $\mathcal{L}(\alpha)$, which equals $abs\_sup(\alpha)$. To obtain the set of frequent 1-patterns, the support of each s-unit u_i is computed from $|\mathcal{L}(u_i)|$ to determine whether it is frequent or not.

For example, the vertical id-list for the grid cells of the dataset $\mathcal{T}_s$ given in Example 5 is shown in Table 9. Each column is a list of occurrences of a cell from the grid $\mathcal{A}_{Grid(G_s)}$ in the given dataset $\mathcal{T}_s$. Consider the grid cell C, C is the fourth s-unit in T_3, the first s-unit in T_6 and the fourth s-unit in T_8, forming the id-list of C. Thus, we have $\mathcal{L}(C) = \{(T_3, 4), (T_6, 1), (T_8, 4)\}$ and $|\mathcal{L}(C)| = 3$ so C is a frequent 1-pattern given $min_{sup} = 0.2$.

Let $join(\mathcal{L}_1 \rightarrow \mathcal{L}_2)$ denote the function that joins id-list $\mathcal{L}_1$ to $\mathcal{L}_2$ on the value of $Trid$ and returns an id-list $\mathcal{L}_{12}$. This join function $\mathcal{L}_{12} = join(\mathcal{L}_1 \rightarrow \mathcal{L}_2)$ performs as follows: For $(trid_1, pid_1) \in \mathcal{L}_1$ and $(trid_2, pid_2) \in \mathcal{L}_2$ such that $trid_1 = trid_2$, if $pid_1 < pid_2$ then add the pair $(trid_2, pid_2)$ to $\mathcal{L}_{12}$.

SPADE applies the lattice theory to structure the search space and traverses

the lattice to enumerate and examine pattern candidates. It generates and tests candidates in the k-iteration as follows:

1. For each pair of $(k-1)$-patterns having the same prefix in S_{k-1}, let say α_1 and α_2, two k-pattern candidates are generated which are $\alpha_{12} = \alpha_1 \rightarrow \alpha_2$ and $\alpha_{21} = \alpha_2 \rightarrow \alpha_1$.

2. Compute $\mathcal{L}(\alpha_{12}) = join(\mathcal{L}(\alpha_1) \rightarrow \mathcal{L}(\alpha_2))$ and $\mathcal{L}(\alpha_{21}) = join(\mathcal{L}(\alpha_2) \rightarrow \mathcal{L}(\alpha_1))$ and test whether α_{12} and α_{21} are frequent by computing the supports from their id-lists $\mathcal{L}(\alpha_{12})$, $\mathcal{L}(\alpha_{21})$.

3. Add a generated pattern to S_k if it is frequent.

We can see that support counting contributes the most to the computation cost of the pattern mining process. Although using id-lists joins is faster than passing over the dataset for support counting, SPADE still suffers an overhead when computing the frequent 2-patterns which either requires N^2 id-list joins for all pair of frequent s-units (N is the cardinality of S_1) or performing a vertical-to-horizontal transformation to perform support counting for 2-pattern candidates in one more scan of the dataset. Also, without considering the MO trajectory characteristics, there can be a lot of unnecessary id-list joins performed by SPADE. This is clearly inefficient when the size of trajectory dataset gets very large. We will address this problem in the next section where we describe an efficient locality-aware approach in our trajectory pattern mining algorithm called TrajPod.

4.4 TrajPod *algorithm*

The key idea of TrajPod is that an id-list of a trajectory pattern can be partitioned into three locality-aware sublists depending on whether its last s-unit occurs in the beginning, the middle or the end of a trajectory. This allows the id-list join involve only the sublists instead of the the whole id-lists. Moreover, by utilizing these

locality-aware sublists, early pruning can be performed effectively which further eliminates trivial patterns to be generated and tested, in addition to using the Apriori pruning principle. This section will describe how TrajPod works in detail. We first introduce the locality-aware partitioned occurrence id-list, called *pod-list*, to track the occurrences of a trajectory pattern. Second, we explain how TrajPod performs id-list joining for support counting. Next, we show how TrajPod enumerates candidate patterns along with its early pruning technique. Finally, we discuss the advantage of using pod-list compared to the simple id-list as used in SPADE.

4.4.1 Locality-aware Partitioned Occurrence id-lists (pod-lists)

MO trajectories tend to overlap a lot due to the fact that mobile objects usually travel following shortest paths. Also, the hotspots in the road network, such as schools, shopping malls, etc. are often endpoints shared by many MO trajectories. We propose a *pod-list* structure for the occurrences of the semantic units in MO trajectories in a road network to capture those facts, which will greatly help speedup the process of forming frequent trajectory patterns.

Definition 23 *(pod-list)* A locality-aware partitioned occurrence id-list (*pod-list*) of a trajectory pattern α, denoted by $\mathcal{L}_{pod}(\alpha)$, is an id-list of α which is partitioned into three disjoint locality-aware sublists: a start list $\mathcal{L}_s(\alpha)$, an intermediate list $\mathcal{L}_m(\alpha)$ and an end list $\mathcal{L}_e(\alpha)$, such that: (i) $\mathcal{L}_{pod}(\alpha) = \mathcal{L}_s(\alpha) \cup \mathcal{L}_m(\alpha) \cup \mathcal{L}_e(\alpha)$, (ii) $\mathcal{L}_s(\alpha) \cap \mathcal{L}_m(\alpha) = \emptyset, \mathcal{L}_m(\alpha) \cap \mathcal{L}_e(\alpha) = \emptyset, \mathcal{L}_e(\alpha) \cap \mathcal{L}_s(\alpha) = \emptyset$, (iii) each pair $(trid, pid) \in \mathcal{L}_{pod}(\alpha)$ is organized into only one of the sublists following these rules in order:

1. $(trid, pid) \in \mathcal{L}_s(\alpha)$ if $pid = 1$
2. $(trid, pid) \in \mathcal{L}_e(\alpha)$ if $pid = |traj(trid)|$
3. $(trid, pid) \in \mathcal{L}_m(\alpha)$ otherwise.

Table 10: Pod-lists of the dataset $\mathcal{T}_s$

Pod-list $\mathcal{L}_{pod}$	Start list $\mathcal{L}_s$	Intermediate list $\mathcal{L}_m$	End list $\mathcal{L}_e$
A	$(T_1, 1), (T_2, 1), (T_4, 1), (T_8, 1)$	$\emptyset$	$\emptyset$
B	$\emptyset$	$(T_1, 2), (T_4, 2), (T_5, 2), (T_8, \{2, 5\})$	$\emptyset$
C	$(T_6, 1)$	$(T_3, 4), (T_8, 4)$	$\emptyset$
D	$(T_7, 1)$	$\emptyset$	$(T_6, 2)$
E	$(T_3, 1), (T_5, 1)$	$(T_4, 3)$	$\emptyset$
F	$\emptyset$	$(T_2, 2), (T_3, 2), (T_8, 3)$	$(T_1, 3), (T_4, 4), (T_5, 3)$
G	$\emptyset$	$(T_2, 3), (T_3, 3), (T_8, 6)$	$\emptyset$
H	$\emptyset$	$\emptyset$	$(T_2, 4), (T_3, 5), (T_8, 7)$

The cardinality $|\mathcal{L}_{pod}(\alpha)|$ equals the number of distinct trajectory identifiers in $\mathcal{L}_s(\alpha) \cup \mathcal{L}_m(\alpha) \cup \mathcal{L}_e(\alpha)$, which is the absolute support of pattern α, i.e., $abs\_sup(\alpha) = |\mathcal{L}_{pod}(\alpha)|$

By definition, $\mathcal{L}_s(\alpha)$ consists of occurrences which are in the form of $(trid, 1)$. Occurrences in $\mathcal{L}_e(\alpha)$ are in the form of $(trid, |traj(trid)|)$ where trajectory $traj(trid)$ has more than one s-unit. Each occurrence $(trid, pid) \in \mathcal{L}_m(\alpha)$ has the value of pid satisfying $1 < pid < |traj(trid)|$.

The first step of TrajPod is to generate the set of pod-lists for all the s-units that occur in the given MO trajectory dataset $\mathcal{T}$. Initially, a pod-list of a s-unit consists of three empty start list, intermediate list and end list. Each trajectory $T_i = \{trid_i, u_1u_2...u_k\}$ of length k is scanned from its first s-unit u_1 to its last s-unit u_k. When scanning an s-unit u_i at the ith position in T_i, we put the pair $(trid, i)$ to one of the sublists in u_i's pod-list $\mathcal{L}_{pod}(u_i)$ based on the rules in the definition of pod-list. For example, while scanning trajectory $T_1 = ABF$, we put $(T_1, 1)$, $(T_1, 2)$, $(T_1, 3)$ into $\mathcal{L}_s(A)$, $\mathcal{L}_m(B)$, $\mathcal{L}_e(F)$ respectively. The pod-lists of the s-units in Example 5 is shown in Table 10.

Definition 24 *(Composite Occurrence)* When there is more than one occurrence of a pattern α with respect to a trajectory T_i in its intermediate list $\mathcal{L}_m(\alpha)$, instead of storing all (T_i, pid_i) pairs representing those occurrences in $\mathcal{L}_m(\alpha)$, we use a compact format of $(T_i, [m, M])$ to represent those occurrences, where m and M are the first and last occurrences of $last(\alpha)$ respectively with respect to T_i in $\mathcal{L}_m(\alpha)$. We call such representation a *composite occurrence*, denoted by $co(T_i, \mathcal{L}_m(\alpha)) = (T_i, [m, M])$.

In case α is a 1-pattern, i.e., $\alpha = u_k$ where u_k is an s-unit, a composite occurrence in its intermediate list is represented by $(T_i, \{m, \ldots, M\})$, where $\{m, \ldots, M\}$ is an ordered list of all the temporal positions of u_k within trajectory T_i, i.e., $\{m, \ldots, M\} = fpos(u_k, T_i)$.

$[m, M]$ or $\{m, \ldots, M\}$ is called the temporal range of the composite occurrence $co(T_i, \mathcal{L}_m(\alpha))$.

For example, given trajectory $T_i = ABCDCACB$, we have $co(T_i, \mathcal{L}_m(AC)) = (T_i, [3, 7])$ and $co(T_i, \mathcal{L}_m(C)) = (T_i, \{3, 5, 7\})$.

Theorem 1 *Given two trajectory pattern α and α' such that $\alpha' \preceq \alpha$ and $last(\alpha') = last(\alpha)$, if α has a composite occurrence with respect to a trajectory T_i in $\mathcal{L}_m(\alpha)$, let say $co(T_i, \mathcal{L}_m(\alpha)) = (T_i, [m, M])$, then α' also has a composite occurrence with respect to a trajectory T_i in $\mathcal{L}_m(\alpha')$ such that $co(T_i, \mathcal{L}_m(\alpha')) = (T_i, [m', M])$ where $m' \leq m$.*

Proof. Since α is a super-pattern of α', any occurrence of α in T_i also contains an occurrence of α'. Given that they share the same last s-unit, any pair (T_i, pid) representing an occurrence of α also represents an occurrence of α', i.e., $fpos(\alpha, T_i) \subseteq fpos(\alpha', T_i)$. Thus, given $co(T_i, \mathcal{L}_m(\alpha)) = (T_i, [m, M])$, α' also has a composite occurrence with respect to a trajectory T_i in $\mathcal{L}_m(\alpha')$, i.e., $co(T_i, \mathcal{L}_m(\alpha')) = (T_i, [m', M'])$ where $m' \leq < m < M \leq M'$. Suppose $last(\alpha') = last(\alpha) = A$, given $co(T_i, \mathcal{L}_m(\alpha)) = (T_i, [m, M])$, by definition of composite occurrence, trajectory T_i must be a sequence of n s-units $u_1 \ldots u_m \ldots u_M \ldots u_n$ $(1 < m < M < n)$ where $u_m = u_M = A$ and $u_i \neq A$ $(\forall u_i \in u_{M+1} \ldots u_{n-1})$. However, we also have $u_{M'} = A$ where $M \leq M' < n$, which enforces $M' = M$. Therefore, we have $co(T_i, \mathcal{L}_m(\alpha')) = (T_i, [m', M])$ where $m' \leq m$.

Definition 25 *(min_pos)* Given a list of integers $\{m, \ldots, M\}$ in ascending order and an integer *pid*, we denote $min\_pos(pid, \{m, \ldots, M\})$ the function that returns the minimum element in $\{m, \ldots, M\}$ which is equal or greater than *pid*.

We call an occurrence represented by $(Trid, pid)$ a *single occurrence* to differentiate it with a composite occurrence. In the rest of the chapter, when we say that an occurrence in a pod-list, i.e., $(Trid, pid) \in \mathcal{L}_{pod}$ or $(Trid, [m, M]) \in \mathcal{L}_{pod}$, we mean that the occurrence belongs to one of the sublists in the pod-list.

4.4.2 Locality-aware Pod-list Join

Suppose we have two $(k-1)$-patterns having the same prefix which are $\alpha_1 = u_1u_2..u_{k-2}a$ and $\alpha_2 = u_1u_2..u_{k-2}b$. The merge operation $\alpha_1 \rightarrow \alpha_2$ forms a k-pattern $\alpha = u_1u_2..u_{k-2}ab$. We will describe how we generate the pod-list of α by joining the pod-lists of α_1 and α_2.

4.4.2.1 Sublist Join

Let $\mathcal{L}_{pod}(\alpha_1) \bowtie \mathcal{L}_{pod}(\alpha_2)$ denote the function that joins the pod-lists $\mathcal{L}_{pod}(\alpha_1)$ to $\mathcal{L}_{pod}(\alpha_2)$ on the value of $Trid$ and returns a pod-list $\mathcal{L}_{pod}(\alpha)$. This join function is performed on the sublists $\mathcal{L}_s$, $\mathcal{L}_m$, $\mathcal{L}_e$ of $\mathcal{L}_{pod}(\alpha_1)$ to those of $\mathcal{L}_{pod}(\alpha_2)$. Joining two sublists is the same as joining two id-lists as used in SPADE, i.e., if there is an occurrence $(trid_1, pid_1)$ in one of a sublist from the left base pattern α_1 and $(trid_2, pid_2)$ in one of a sublist from the right base pattern α_2 such that $trid_1 = trid_2$, if $pid_1 < pid_2$ then the pair $(trid_2, pid_2)$ will be added to the corresponding sublist in $\mathcal{L}_{pod}(\alpha)$. Here, when performing an equal join on the value of $trid$, we only check the inequality $pid_1 < pid_2$ within a trajectory because a trajectory is a sequence of one-item events, in which a has to temporally precede b if there exists pattern $\alpha = u_1u_2..u_{k-2}ab$ in that trajectory. Since we perform the merge $\alpha_1 \rightarrow \alpha_2$ to generate α, depending on the locality property of the sublists on each side of the merge operator, we note the following observations when joining the sublists:

- Since there is only one s-unit at the beginning of each trajectory, i.e., an occurrence in a start list has to be in the form of $(trid, 1)$, there can not exist

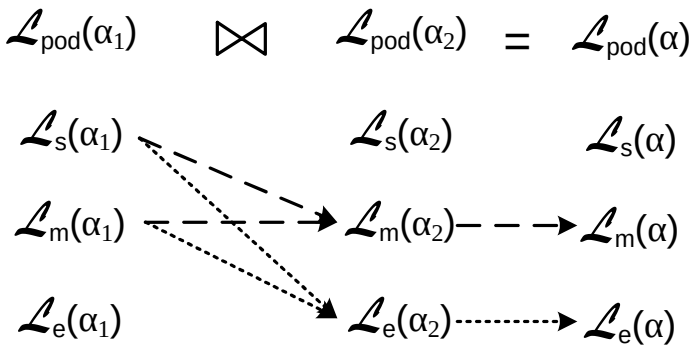


Figure 31: Locality-aware Pod-list Join

an inequality of *pid* assuming the same *trid* from $\mathcal{L}_s(\alpha_1)$ and $\mathcal{L}_s(\alpha_2)$. In another words, an occurrence of an s-unit at the beginning of a trajectory can not be preceded by any other occurrence. Therefore, the start list of the left base pattern α_1 can not be joined to start list of the right base pattern α_2.

- An occurrence of an s-unit at the end of a trajectory can not be followed by any other occurrence. Thus, the end list of the left base pattern α_1 can not be joined to any of the three sublists of the right base pattern α_2.

Based on those observations, instead of performing 9 sublist joins from $\mathcal{L}_s(\alpha_1)$ to $\mathcal{L}_s(\alpha_2)$, there are only four possible sublist joins to produce $\mathcal{L}_{pod12}$ as depicted in Figure. 31 including:

- $\mathcal{L}_s(\alpha_1) \bowtie \mathcal{L}_m(\alpha_2)$, called $(s \bowtie m)$ join, produces the occurrences of pattern α to be added to its intermediate list $\mathcal{L}_m(\alpha)$.

- $\mathcal{L}_s(\alpha_1) \bowtie \mathcal{L}_e(\alpha_2)$, called $(s \bowtie e)$ join, produces the occurrences of pattern α to be added to its end list $\mathcal{L}_e(\alpha)$.

- $\mathcal{L}_m(\alpha_1) \bowtie \mathcal{L}_m(\alpha_2)$, called $(m \bowtie m)$ join, produces the occurrences of pattern α to be added to its intermediate list $\mathcal{L}_m(\alpha)$.

- $\mathcal{L}_m(\alpha_1) \bowtie \mathcal{L}_e(\alpha_2)$, called $(m \bowtie e)$ join, produces the occurrences of pattern α to be added to its end list $\mathcal{L}_e(\alpha)$.

From these joining rules, we can see that a generated pattern always has an empty start list. This can also be seen from the definition of pod-list such that in a k-pattern where $k > 1$, its last s-unit is always be preceded by at least one s-unit. Thus the value of *pid* corresponding to the occurrence of that pattern in a trajectory can not be 1, making its start list empty. Therefore, when generating candidates from base patterns with lengths that are greater than 1, we only use their intermediate lists and end lists, which means we only have to perform two sublist joins ($m \bowtie m$ join and $m \bowtie e$ join) to create the pod-list for each pattern generated. The s-m join and s-e join are only performed for joining pod-lists of frequent s-units, which are frequent 1-patterns, as base patterns.

4.4.2.2 Join on composite occurrences

We show how to perform a temporal comparison on the value of *pid* when there is a composite occurrence involved in a pod-list join. The purpose of using a composite occurrence is to reduce data space as well as to reduce the number of temporal comparisons on *pid* in case of a matching *trid* compared to using single occurrences only. Composite occurrences can involve a pod-list join in these following three cases:

- In an $m \bowtie m$ join or $m \bowtie e$ join, there can be $(trid_1, [m_1, M_1]) \in \mathcal{L}_m(\alpha_1)$ and $(trid_2, pid_2) \in \mathcal{L}_{m|e}(\alpha_2)$ such that $trid_1 = trid_2 = T_i$. We only need to make a temporal comparison for m_1 and pid_2, if $m_1 < pid_2$ which means pattern $\alpha = \alpha_1 \rightarrow \alpha_2$ occurs in trajectory $traj(T_i)$, thus, we add the single occurrence $(trid_2, pid_2)$ to $\mathcal{L}_m(\alpha)$ or $\mathcal{L}_e(\alpha)$ accordingly.

- In an $m \bowtie m$ join, there can be $(trid_1, pid_1) \in \mathcal{L}_m(\alpha_1)$ and $(trid_2, [m_2, M_2]) \in \mathcal{L}_m(\alpha_2)$ such that $trid_1 = trid_2 = T_i$. We only need to make a temporal comparison for $trid_1$ and M_2, if $pid_1 < M_2$ which means pattern $\alpha = \alpha_1 \rightarrow \alpha_2$ occurs in trajectory $traj(T_i)$.

  - If $pid_1 \leq m_2$, we add $(T_i, [m_2, M_2])$ to $\mathcal{L}_m(\alpha)$.

- If $pid_1 > m_2$, according to Theorem 1, the intermediate list of $last(\alpha_2)$ has a composite occurrence $(T_i, \{m, \ldots, M_2\})$ where $m \leq m_2$. We compute $m_2' = min\_pos(pid_1, \{m, \ldots, M_2\})$ and add the composite occurrence $(T_i, [m_2', M_2])$ to $\mathcal{L}_m(\alpha)$.

- In an $m \bowtie m$ join, there can be $(trid_1, [m_1, M_1]) \in \mathcal{L}_m(\alpha_1)$ and $(trid_2, [m_2, M_2]) \in \mathcal{L}_m(\alpha_2)$ such that $trid_1 = trid_2 = T_i$. Pattern $\alpha = \alpha_1 \rightarrow \alpha_2$ only occurs in trajectory $traj(T_i)$ when $m_1 < M_2$. In this case, similar to the above case, we only need to check whether m_1 falls in or out of the range $[m_2, M_2]$ to compute the occurrences of α in T_i.

  - If $m_1 < m_2 < M_2$, we add $(T_i, [m_2, M_2])$ to $\mathcal{L}_m(\alpha)$.
  - If $m_2 < m_1 < M_2$, given $(T_i, \{m, \ldots, M_2\})$ as the composite occurrence of $last(\alpha_2)$ in its intermediate list, we compute $m_2' = min\_pos(m_1, \{m, \ldots, M_2\})$ and add the composite occurrence $(T_i, [m_2', M_2])$ to $\mathcal{L}_m(\alpha)$.

Theorem 2 *Given two k-patterns α_1 and α_2, which are the left base pattern and right base pattern of a pattern α respectively. Let $|\mathcal{L}_s(\alpha_1) \cup \mathcal{L}_m(\alpha_1)|$ and $|\mathcal{L}_m(\alpha_1)|$ denote the number of distinct trid in $\mathcal{L}_s(\alpha_1) \cup \mathcal{L}_m(\alpha_1)$ and $\mathcal{L}_m(\alpha_1)$ respectively. Let $|\mathcal{L}_m(\alpha_2) \cup \mathcal{L}_e(\alpha_2)|$ denote the number of distinct trid in $\mathcal{L}_m(\alpha_2) \cup \mathcal{L}_e(\alpha_2)$. We have $|\mathcal{L}_{pod}(\alpha)| \leq |\mathcal{L}_s(\alpha_1) \cup \mathcal{L}_m(\alpha_1)|$ if $k = 1$, and $|\mathcal{L}_{pod}(\alpha)| \leq |\mathcal{L}_m(\alpha_1)|$ if $k > 1$. We also have $|\mathcal{L}_{pod}(\alpha)| \leq |\mathcal{L}_m(\alpha_2) \cup \mathcal{L}_e(\alpha_2)|$.*

Proof. According to the pod-list joining rules, if $k = 1$, only the start list and the intermediate list of the left base pattern α_1 take part in the join to produce $\mathcal{L}_{pod}(\alpha)$. If $k > 1$, only the intermediate list of α_1 takes part in the join to produce $\mathcal{L}_{pod}(\alpha)$. Also, only the intermediate list and the end list of the right base pattern α_2 take part in the join to produce $\mathcal{L}_{pod}(\alpha)$. In a pod-list join, the number of distinct *trid*, i.e., the cardinality of the resulting pod-list can not exceed the participating cardinality from either side of the $\bowtie$ operator. Therefore, we have the theorem.

4.4.3 Candidate Pattern Generation and Test

In this section, we present the pattern tree structure that covers the complete search space of the trajectory pattern mining problem and how TrajPod traverses the pattern tree and computes the supports using the pod-list join operation to output frequent patterns.

Definition 26 *(Trajectory Pattern Tree)* Given a set $\mathbb{U}$ of s-units associated with a MO trajectory dataset $\mathcal{T}$, a trajectory pattern tree, denoted by $tree_{\mathbb{U}}(\mathcal{T})$, is a finite tree structure having these following properties:

- Each node is a trajectory pattern comprised of the s-units in $\mathbb{U}$
- Trajectory patterns of the same length are arranged in the same level in the tree
- A trajectory pattern at a parent node is the prefix of all its child nodes
- The maximum height of the tree equals the maximum length of the trajectories in $\mathcal{T}$

A trajectory pattern tree $tree_{\mathbb{U}}(\mathcal{T})$ starts with an empty set $\emptyset$ at its root, i.e., level 0. All the s-units in $\mathbb{U}$ can be at level 1 as 1-patterns. At level k ($k > 1$), the children of a node p can be generated given all siblings of p. Let $sib(p)$ denote the set consisting of all siblings of node p and p itself. All patterns in $sib(p)$ share the same prefix. Node p will act as the left base pattern to merge with a pattern in $p' \in sib(p)$. The merge operation $p \rightarrow p'$ will return a child of node p.

Obviously, a pattern tree $tree_{\mathbb{U}}(\mathcal{T})$ completely covers the search space of the trajectory pattern mining problem given a trajectory dataset $\mathcal{T}$. However, it is not necessary and too costly to generate and test all the patterns in a tree. Since the Apriori property holds, if a pattern is trivial, then all of its children are also trivial.

Thus, when the support of a pattern at a node is less than *min_sup*, we do not need to store it in the tree and stop the expansion from that node on.

4.4.3.1 Early Pruning

Even when a pattern at a node is frequent, there might also be no need to generate and test its children, thanks to the use of pod-list in TrajPod. Specifically, TrajPod tags each pattern at a node with its pod-list, where we can check the *inextensibility* of a node p from the cardinality of $\mathcal{L}_{pod}(p)$'s sublists.

Definition 27 *(Inextensible pattern)* Given a k-pattern α, let $|\mathcal{L}_s(\alpha) \cup \mathcal{L}_m(\alpha)|$ and $|\mathcal{L}_m(\alpha)|$ denote the number of distinct *trid* in $\mathcal{L}_s(\alpha) \cup \mathcal{L}_m(\alpha)$ and $\mathcal{L}_m(\alpha)$. We say that α is inextensible if $|\mathcal{L}_s(\alpha) \cup \mathcal{L}_m(\alpha)| < abs\_minsup$ with $k = 1$ or $|\mathcal{L}_m(\alpha)| < abs\_minsup$ with $k > 1$.

Because a k-pattern α at a node will act as the left base pattern when generating a child β, if α is *inextensible*, according to Theorem 2, we have $|\mathcal{L}_{pod}(\beta)| < |\mathcal{L}_s(\alpha) \cup \mathcal{L}_m(\alpha)| < abs\_minsup$ for $k = 1$ or $|\mathcal{L}_{pod}(\beta)| < \mathcal{L}_m(\alpha)| < abs\_minsup$ for $k > 1$. This shows that if a node is inextensible, its children must be trivial. Therefore, we do not need to expand the trajectory pattern tree from an inextensible node.

We denote $ip\_check(\alpha)$ as a function that checks if a pattern α is inextensible or not. $ip\_check(\alpha)$ returns *true* if α is inextensible and returns *false* otherwise.

In addition, let u_1 and u_2 be two 1-patterns at level 1 of the pattern tree. When we perform the merge $u_1 \rightarrow u_2$ to produce the 2-pattern u_1u_2 which is a child of u_1, only the intermediate list $\mathcal{L}_m(u_2)$ and the end list $\mathcal{L}_e(c_u)$ participate in the right side of the join $\mathcal{L}_{pod}(u_1) \bowtie \mathcal{L}_{pod}(u_2)$ to produce $\mathcal{L}_{pod}(u_1u_2)$. If the cardinality of $|\mathcal{L}_m(u_2) \cup \mathcal{L}_e(u_2)|$ is less than abs_minsup then according to Theorem 2, $|\mathcal{L}_{pod}(u_1u_2)| < abs\_minsup$, which makes u_1u_2 trivial. Such an s-unit is called an *unmergeable pattern* and is formally defined in the following definition.

Definition 28 *(Unmergeable pattern)* Given an s-unit u, we denote $r\_check(u)$ as a function that returns *true* if $|\mathcal{L}_m(u) \cup \mathcal{L}_e(u)| < abs\_minsup$ and returns *false* otherwise. If $r\_check(u)$ is true, u is called an *unmergeable* pattern and it should not be chosen to be a right base pattern in a merge when producing a 2-pattern at level 2 of the pattern tree.

The sublist cardinality check techniques, *ip_check* and *r_check*, allows early pruning in TrajPod where unnecessary candidate generations and tests are eliminated.

4.4.3.2 Depth First Traversal

TrajPod first computes the support for each s-unit based on its pod-list to produce the set of frequent 1-patterns S_1, which is then added to level 1 of $tree_{\mathbb{U}}(\mathcal{T})$. TrajPod spans the trajectory pattern tree in a depth-first fashion. When visiting a node p, TrajPod will check the inextensibility of p using *ip_check*. If the node is not subjected to early pruning, TrajPod will generate all the child patterns of p along with their pod-lists and compute their supports based on the post-list cardinalities. A node at level 1 is also checked if it is unmergebale using *r_check* before being considered as a right base pattern. Only when the support of a child pattern is equal or greater than *min_sup*, it is kept in the tree and recursively depth-first expanded. The traversal finishes when TrajPod completes visiting all the nodes at the tree's maximum level or it reaches all the nodes with trivial patterns. Algorithm 7 shows the pseudo-code for TrajPod's depth-first visit to a node p in the pattern tree $tree_{\mathbb{U}}(\mathcal{T})$.

A sample trajectory pattern tree for the dataset $\mathcal{T}_s$ in our running example is shown in Figure 32. All trivial nodes, which are marked with an overline in the figure, are pruned out according to the Apriori policy. By performing id-sublist cardinality checks, we can detect the *inextensible patterns*, marked with a circle cover, where we can stop expanding the tree. We can also detect the *unmergebale* 1*-patterns*,

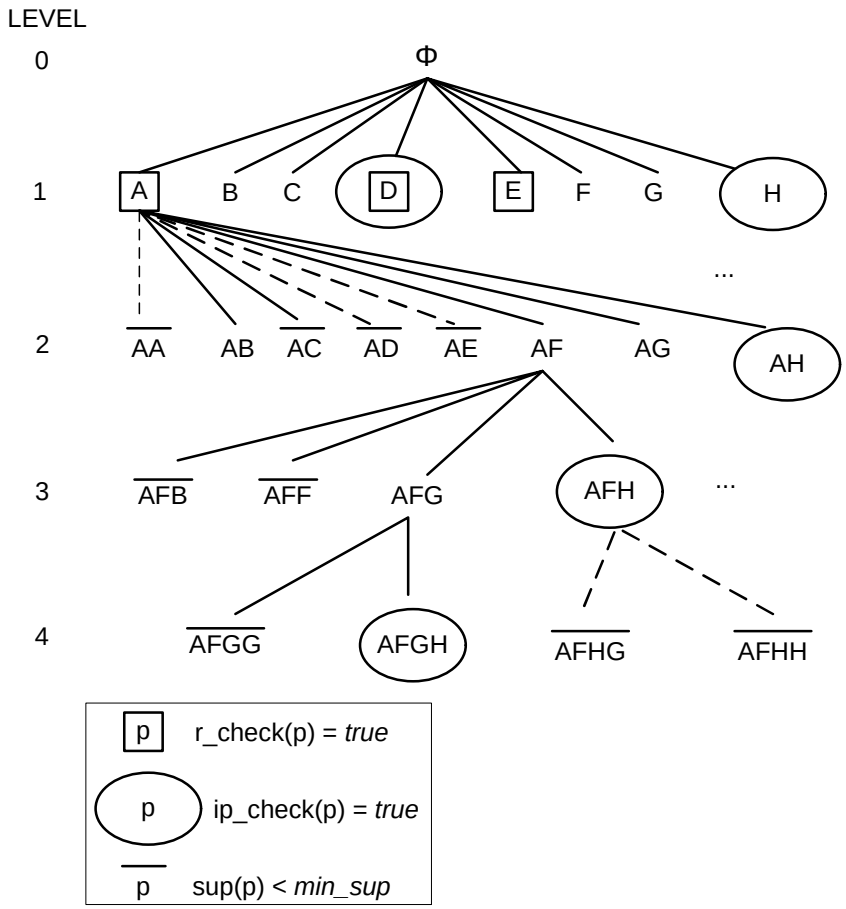


Figure 32: The trajectory pattern tree $tree_{\mathbb{U}}(\mathcal{T}_s)$

marked with a rectangle cover, which are incapable of being merged into 2-patterns. The dotted-line edges describe the unnecessary expansions which are pruned early by using the id-sublist cardinality checks in TrajPod. For example, there are 8 frequent s-units at level 1. Given $min\_sup = 0.25$, i.e., $abs\_minsup = 2$, we have D is inextensible since $|\mathcal{L}_s(D) \cup \mathcal{L}_m(D)| = 1 < abs\_minsup$. D is also unmergeable since $|\mathcal{L}_m(D) \cup \mathcal{L}_e(D)| = 1 < abs\_minsup$. Similarly, we can determine that only A, B, C, E, F, G can be left base 1-patterns and only B, C, F, G, H can be right base 1-patterns. Thus, instead of performing $8 \times 8 = 64$ SPADE-like id-list joins among S_1, TrajPod only needs to perform $6 \times 5 = 30$ pod-list joins.

Algorithm 7 DFS($tree_{\mathbb{U}}(\mathcal{T})$, node p, min_sup)

```
1: k = level(p)
2: r_check = false
3: for p' ∈ sib(p) do
4:    if k = 1 then
5:       r_check = r_check(p')
6:    end if
7:    if (r_check = false) then
8:       p_i = p → p'
9:       L_pod(p_i) = L_pod(p) ⋈ L_pod(p')
10:      if sup(p_i) ≥ min_sup then
11:         insert p_i into children(p)
12:         add p_i to S_{k+1}
13:      end if
14:   end if
15: end for
16: for p_i ∈ children(p) do
17:   if (ip_check(p_i) = false) then
18:      DFS(tree_U(T), node p_i, min_sup)
19:   end if
20: end for
```

4.4.4 Pod-list vs. id-list

We can see the advantages of using pod-list to represent MO trajectories compared to the naïve id-list as used to represent traditional sequence data in SPADE. The cardinalities of both pod-lists and id-lists shrink and thus, making the joins run faster as the patterns extend their lengths. However, since the id-list cardinalities of k-patterns with small k, e.g., $k = 1, 2$, are the largest during the mining process, computing the corresponding set S_{k+1} becomes a bottleneck for SPADE. For example, computing the set of frequent 2-patterns S_2 from S_1 using the id-lists requires N^2 joins ($N = |S_1|$) of large id-lists, which is very inefficient. That is why SPADE has to either make an additional scan of the database to perform a vertical-to-horizontal transformation for the frequent items and has to count the support of the 2-pattern candidates, or use a preprocessing step that computes the counts of all 2-patterns. In contrast, TraPod effectively uses pod-list throughout its mining process. To

compute S_2, TraPod only needs to perform $m \times n$ pod-list joins where m is the number of s-units passing *ip_check* and n is the number of s-units passing *r_check*. TraPod is fast because (i) the pod-list join is sublist joining instead of joining the whole id-lists, (ii) it is able to perform early pruning using the id-sublist cardinality check techniques which are *ip_check* and *r_check*, and (iii) it implements the generate and test framework in a depth first search fashion which is memory-efficient.

4.5 Experimental Evaluation

We present three sets of experimental results to demonstrate the efficiency and robustness of TrajPod. First, we show the performance comparison in terms of running times and memory usage of TrajPod with state-of-the-art sequential pattern mining algorithms SPADE, SPAM and PrefixSpan. Second, we analyze the performance of TrajPod on grid-based trajectory datasets of varying grid cell sizes in different road networks. Third, we test the scalability of TrajPod for varying size trajectory datasets of both grid-based and segment-based s-units.

4.5.1 Experimental Setup

Road Networks. We use three real road networks of different scales of the road network topologies in terms of geometry and spatial density in our experiments (Table 11). The road networks of North West Atlanta (ATL) of rural/suburban style and West San Jose (SJ) of urban style are obtained from [85]. The Miami-Dade (MIA) road network of urban style is obtained from [83]. MIA map is an high-density urban road network with a dense, regular grid structure, short streets, and most intersections with four connecting streets. SJ map has similar structure but less dense. ATL map has the least density in terms of network topology. We modify the public event-based simulator GTMobiSIM [70] to generate thousands of mobility traces on those road networks for a large-scale evaluation.

Table 11: Real road networks used in our experiments

Style	Regions	Total length	# Segments	# Junctions	Avg. segment length	Junction degree
suburban	North West Atlanta, GA	1384.4km	9187	6979	150.7m	avg: 2.6, max: 6
suburban	West San Jose, CA	1821.2km	14600	10929	124.7m	avg: 2.7, max: 6
urban	Miami-Dade, FL	26148.3km	154681	103377	169.0m	avg: 3.0, max: 9

Datasets. To generate trajectories in a given road network as in real life traveling, we place mobile objects in the road networks with speed limit constraints on road segments, following the shortest paths from their beginning location to their destination. The start and end locations in each trip of a mobile object are randomly chosen from a predefined set of hot spots on the map. Each object moves independently of others, under the speed limit and speed distribution defined for the road segments. The speed limits are set for each type of road at 30mph for residential, 55mph for highway, 70mph for freeway and 30mph for freeway interchange. Object speeds are chosen from a Gaussian distribution with a standard deviation of 0.2 times the mean, which is the road segment speed limit. The road network locations of an object are recorded every 5 seconds during its trip. The simulated road network trajectories are then transformed into sequences of semantic spatial units, which are either grid cells or road segments, to be the input sequences for our trajectory pattern mining algorithm.

The naming of a dataset is in the form of $(MAP).$(UNIT).T$NK in which $(MAP) is the name code of the road network (ATL, SJ or MIA), $(UNIT) is the mode of s-unit for MO trajectories which is either GRID of SEG (GRID for grid cell or SEG for road segment) and $NK is the number of trajectories which is a multiple of thousands. For example, SJ.GRID.T10K is a trajectory dataset consisting of 10000 trajectories as grid cell sequences in West San Jose road networks.

Choice of Cell Sizes. The cell size $\alpha \times \beta$ determines the granularity of the spatial proximity of road network locations in a road network $G = (\mathcal{V}, \mathcal{E})$. When using the grid cells as the semantic spatial units, a user can provide their desirable cell size or use the suggested cell size generated by the system to determine the grid

structure of a road network. In the first prototype of TrajPod, we determine the cell size based on the average number of road segments, computed by $\frac{|\mathcal{E}|}{M \times N}$, where $|\mathcal{E}|$ is the total road segments of the road network $G = (\mathcal{V}, \mathcal{E})$. Given that segments are not uniformly distributed across a road network, the number of road segments per cell should be within an appropriate range $[n_l, n_u]$ such that $n_l \leq \lfloor \frac{|\mathcal{E}|}{M \times N} \rfloor \leq n_u$, n_l and n_u are system supplied parameters. We use a quad grid of size $2^m \times 2^n$. Given that a cell area defines the spatial proximity in the road network, we suggest that a grid cell should cover the portions from an appropriate number of road segments. For example, if we choose a grid cell to cover an average of 1 to 3 road segments, then m can be determined to satisfy the condition of $1 \leq \lfloor \frac{|\mathcal{E}|}{2^m \times 2^n} \rfloor \leq 3$. In our experiments, the cell size is by default set to $[186m \times 220m]$ and $[176m \times 218m]$, which are equivalent to the same grid size of $[2^6 \times 2^6]$, for ATL and SJ maps respectively, so that the average number of road segments residing in a grid cell is in the range of $[1, 3]$. The grid size for MIA road network is set to $[2^9 \times 2^9]$ due to the larger scale of MIA map, which is approximately equivalent to the use of cell size of $[180m \times 184m]$.

All algorithms are written in Java and take as input trajectory data sequence format. The SPADE and SPAM implementations are conversions of C++ code obtained from M. J. Zaki [78] and J. Ayres [79] to Java programs respectively. When running the experiments, the frequent trajectory patterns produced by the algorithms are not saved to ensure only their execution times are recorded. All the experiments are conducted on a computer with Intel Core i7 CPU of 2.40GHz and 8GB of main memory.

4.5.2 Performance Comparison

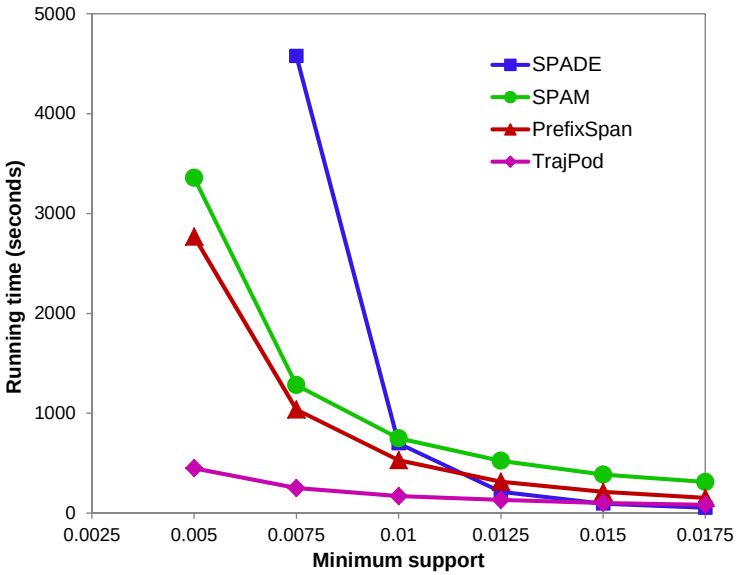
In this set of experiments, we study the relative performance of four algorithms TrajPod, SPADE, SPAM and PrefixSpan on trajectory sequences of both grid-based s-units and segment-based s-units for three different-scale maps with varying

minimum support thresholds. Both running times and memory usage of each algorithm are reported in each test to better evaluate its efficiency. Memory usage is measured by the maximum memory utilization at a point of time during the algorithm execution.

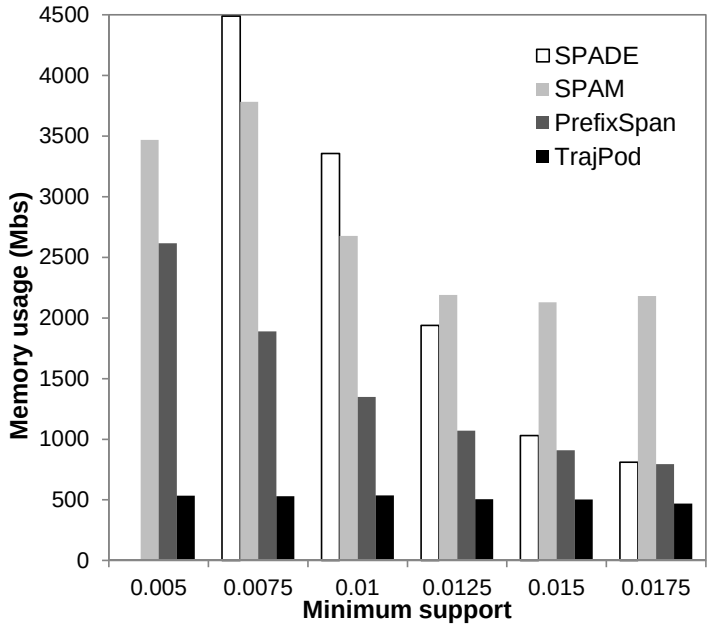
Results for ATL map, a rural/suburban road network. The performance comparison for ATL.SEG.T10K dataset is reported in Figure 33. When $min\_sup \in \{0.0175, 0.015, 0.0125\}$, the number of frequent patterns in the dataset is small. SPADE is marginally faster than other algorithms since the id-list creation and the naïve id-list joining cause the least overhead compared to other approaches. TrajPod runs faster than SPAM and Prefix and runs a bit slower than SPADE. As *min_sup* drops, SPADE becomes the least efficient. It is because when the number of frequent patterns increases, SPADE's simplicity is no longer an advantage. SPADE has to perform a lot more id-list joins and starts having a surge in memory usage when $min\_sup = 0.0075$ as shown in Figure 33(b). When *min_sup* drops to 0.005, SPADE could not finish. SPAM consumes a lot of memory due to the space-inefficient bitmap representations and runs slower than PrefixSpan. The overhead of the bitmap manipulation in SPAM and projected database generation in PrefixSpan increase as *min_sup* decreases, while TrajPod consumes much less memory and remains relatively small running times. TrajPod runs about 7 times faster than Prefix and SPAM at $min\_sup = 0.005$.

The result for ATL.GRID.T10K dataset is shown in Figure 34. SPADE is the slowest and fails to run when *min_sup* drops below 0.015. The running time of SPAM is a bit less than TrajPod when $min\_sup \geq 0.02$ but it grows significantly as *min_sup* drops below 0.002. Overall, PrefixSpan is slower than SPAM and TrajPod is the most efficient. It is because TrajPod smartly utilizes trajectory locality to perform early and deep candidate pruning and locality-aware sublist joins, which significantly reduces the search space and speedups the candidate testing process.

Results for SJ map, an urban road network. The performance reports for



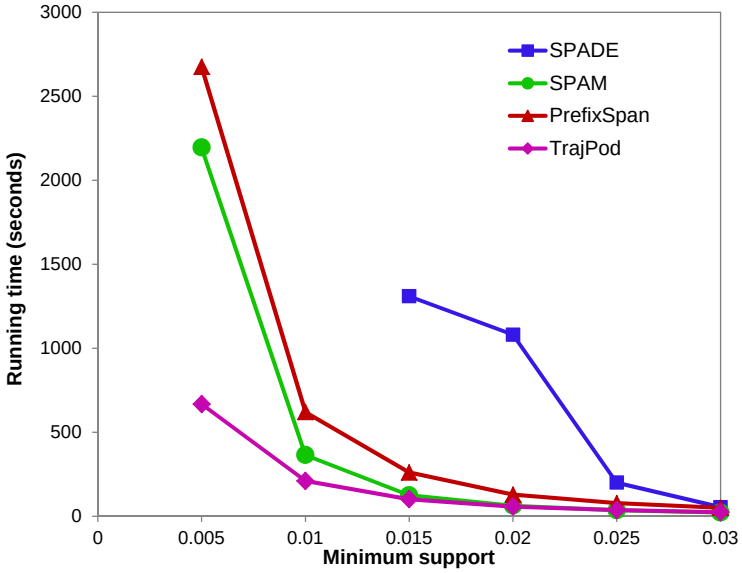
(a) Running times



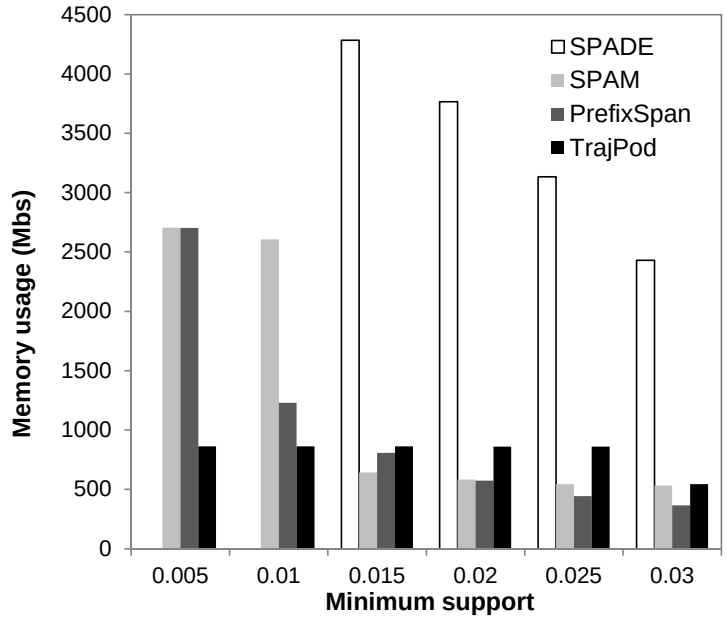
(b) Memory Usage

Figure 33: Performance comparison for ATL.SEG.T10K

datasets of West San Jose road network plotted in Figure 35 and Figure 36 confirm the efficiency in mining MO trajectory patterns of TrajPod. The result for SJ.GRID.T10K dataset is similar to ATL.GRID.10K dataset. In the resulting graph for SJ.SEG.T10K dataset, the overall order in running times is TrajPod < SPAM



(a) Running times



(b) Memory Usage

Figure 34: Performance comparison for ATL.GRID.T10K

< SPADE < PrefixSpan. Table 12 reports the number of frequent trajectory patterns at varying minimum supports in our test trajectory datasets of ATL and SJ road networks. Although the number of frequent patterns in SJ.SEG.T10K dataset increases much faster as *min_sup* drops and is significantly larger than that of other datasets at the low *min_sup* value of 0.005, TrajPod still runs very smoothly. TrajPod is

Table 12: Total number of frequent trajectory patterns

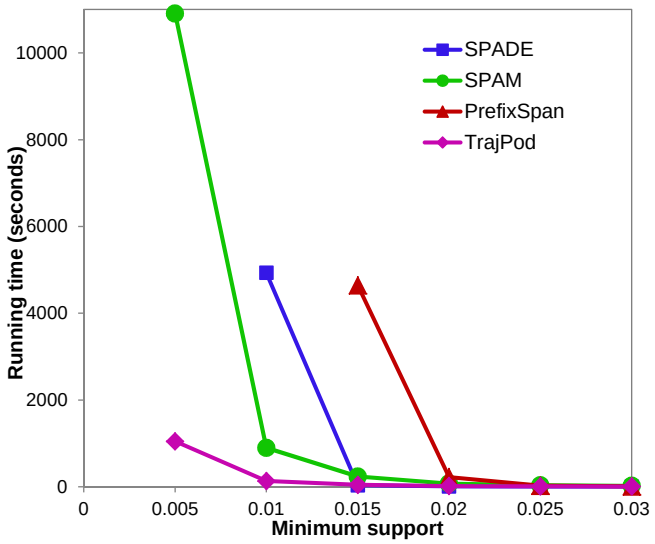
Datasets	*min_sup*		
	0.005	0.01	0.015
ATL.SEG.T10K	814837	74507	18329
SJ.GRID.T10K	2948493	245971	57237
ATL.GRID.T10K	15883775	1505980	348812
SJ.SEG.T10K	40739992	294694	20207

Table 13: Segment-based Trajectory Datasets

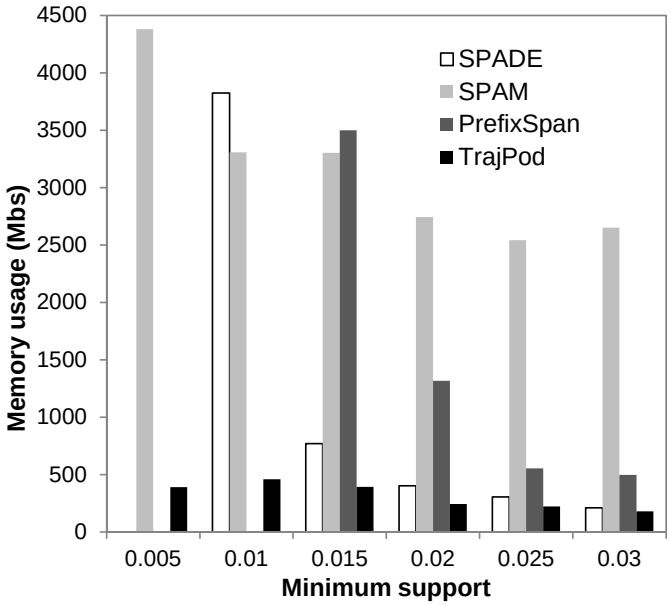
Datasets	**trajectory lengths**			**# *s*-units**
	min	max	mean	
ATL.SEG.T10K	31	374	111	7998
SJ.SEG.T10K	49	341	116	13170
MIA.SEG.T10K	34	970	420	60480

an order of magnitude faster than SPAM at $min\_sup = 0.005$, while both PrefixSpan and SPADE never terminate when *min_sup* drops to 0.02 and 0.015 respectively. We also see the correlation between memory usage and running time in each test. When the memory is utilized efficiently, as in the execution of TrajPod, the running time is relatively stable. In the execution of other algorithms, when the memory is nearly exhausted, the algorithm runs very slow and eventually fails to finish.

Results for MIA map, a high-density urban road network. The performance measurement for MIA.SEG.T10K dataset displayed in Figure 37(a) clearly shows that TrajPod is a winner over SPAM, SPADE and PrefixSpan. Since MIA map is a high density road network, trajectories in the form of sequence of road segments are longer in terms of *s*-unit cardinality. Also, the number of *s*-units, i.e., road segments, across a MIA.SEG dataset is larger than that of a ATL.SEG dataset and a SJ.SEG dataset. We can see these differences from Table 13 showing the statistics of three segment-based trajectory datasets of ATL, SJ and MIA maps that we use in our experiments. SPAM could not run and throws an error



(a) Running times

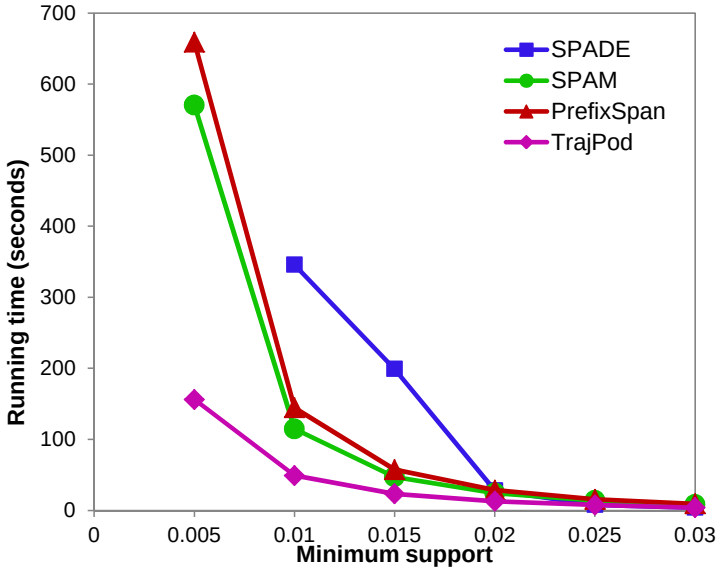


(b) Memory Usage

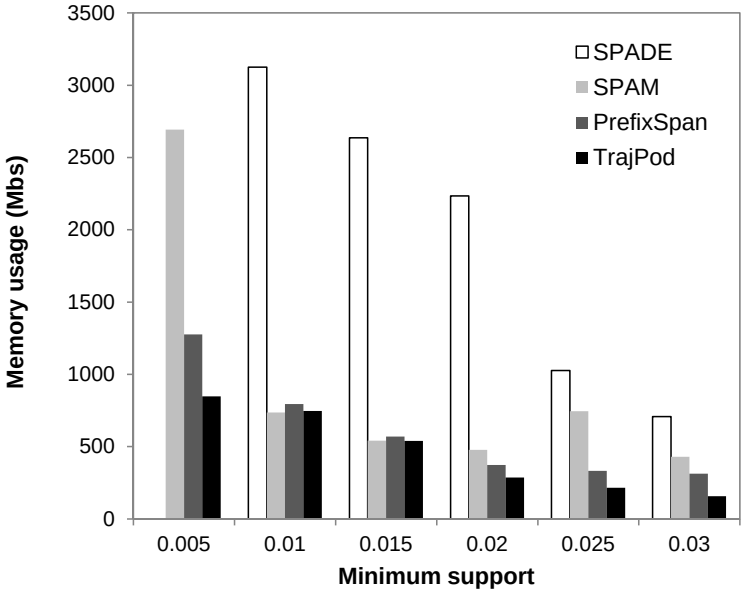
Figure 35: Performance comparison for SJ.SEG.T10K

"java.lang.OutOfMemoryError" for all values of *min_sup*. SPADE is relatively competitive to TrajPod but could not terminate when *min_sup* drops to 0.04. PrefixSpan consumes the most memory, which can be due to the large prefix-projected databases it has to create and examine recursively, and runs the slowest.

Regarding efficiency, SPAM is extremely space-inefficient since the bitmap vertical



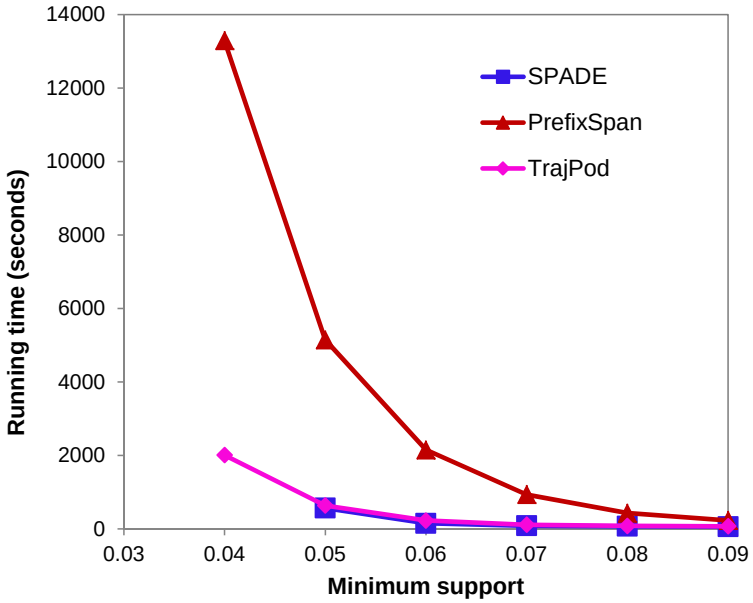
(a) Running times



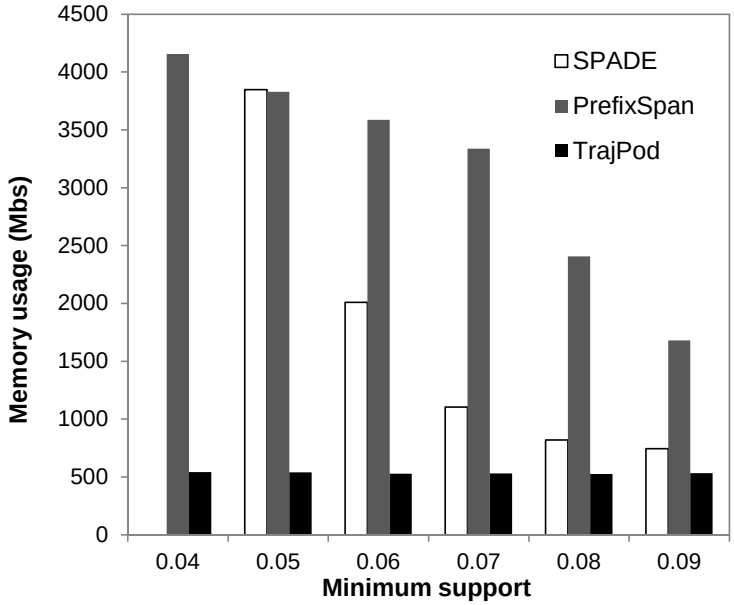
(b) Memory Usage

Figure 36: Performance comparison for SJ.GRID.T10K

layout of a dataset consumes much larger space than the original size of the dataset. For each s-unit, its bitmap requires one bit for every position in each trajectory. Given D trajectories with an average s-unit cardinality of C and U is the number of distinct s-units across all the trajectories, SPAM requires $(D \times C \times U)/8$ bytes to store all the data. While in the vertical layout of the dataset in SPADE uses $D \times C \times 2$ bytes to



(a) Running times



(b) Memory Usage

Figure 37: Performance comparison for MIA.SEG.T10K

record $D \times C$ s-unit occurrences (assuming it costs 2 bytes to store each occurrence). Usually, $C \ll U$ (as $C = 420$ compared to $U = 60480$ in MIA.SEG.T10K dataset shown in Table 13). TrajPod saves more space than SPADE by using the compact representation of composite occurrence. The additional space that TrajPod

needs to mark three sublists within a pod-list is negligible. SPADE, although is more space-efficient in the beginning, suffers from the level-wise generation of many candidates and naïve id-list joining without any pruning and computational optimization. Especially during the first several iterations, when the number of frequent $(k-1)$-patterns is large, SPADE can experience a bottleneck dealing with joining the large number of big-size id-lists of $(k-1)$-patterns to find frequent k-patterns. The distribution of frequent trajectory patterns of MIA.SEG.T10K dataset is shown in Figure 38. We can see that at $min\_sup = 0.04$, the number of 2-patterns significantly increases, causing bottleneck for SPADE. PrefixSpan has major cost in generating and testing projected databases recursively which can cause bottleneck when the number of projected databases does not shrink well. In contrast, TrajPod is specifically designed for trajectory data with effective mechanisms for deep pruning and fast support counting. Although other existing approaches in some cases are marginally faster than TrajPod when the values of min_sup are high, TrajPod overall outperforms them in terms of running time and memory usage with different scales of maps and varying minimum support thresholds.

4.5.3 Varying Grid Cell Sizes

We measure the performance of TrajPod by varying the settings of grid cell size for all three road networks ATL, SJ and MIA. We run TrajPod on the datasets of 10K grid-based trajectories on each road network. Figure 39 reports the running times of TrajPod for different settings of cell sizes, from $[372m \times 440m]$ down to $[93m \times 110m]$. The value of min_sup is set to 0.008 in this experiment. When the cell size gets smaller, a MO trajectory will be presented by a sequence of larger number of grid cells. The increased s-unit cardinality of a MO trajectory results in the increased number of s-units in a grid-based trajectory dataset as well as the increased number of trajectory patterns. Therefore, it takes more time for TrajPod

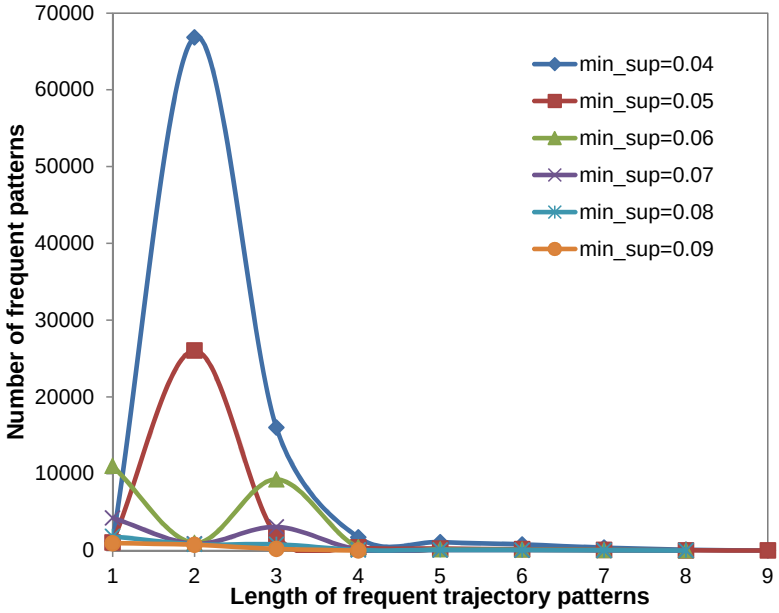


Figure 38: The distribution of frequent trajectory patterns of MIA.SEG.T10K dataset.

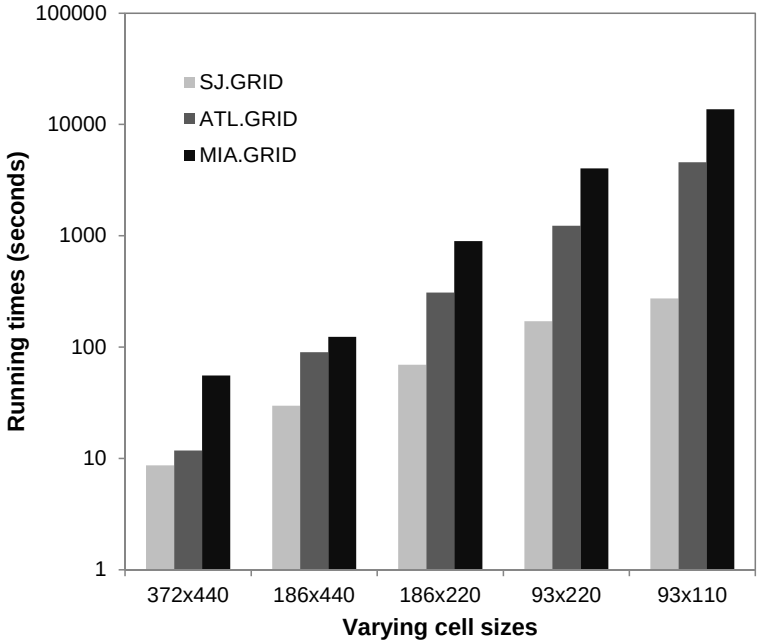


Figure 39: Running times of TrajPod vs. varying grid cell sizes

to finish running as the cell size associated with a map decreases. This is confirmed by the results shown in Figure 39 that the running time increases as the cell size decreases for each map. We can also see that with the same cell size, it takes the

Table 14: Grid-based Trajectory Datasets, cell size = $[186m \times 220m]$

Datasets	**trajectory lengths**			**# s-units**	**# patterns**
	min	max	mean		
ATL.GRID.T10K	9	59	32	2457	3274546
SJ.GRID.T10K	4	64	34	3167	547343
MIA.GRID.T10K	9	162	81	14181	537308

longest for TrajPod to finish on MIA map compared to ATL and SJ maps. This can be attributed to the larger scale of MIA map compared to the other maps. As shown in Table 14, with the same cell size of $[186m \times 220m]$, the mean length of trajectories in terms of number of grid cells of MIA.GRID.T10K dataset is about 2.4 times longer than that of SJ.GRID.T10K dataset and ATL.GRID.T10K dataset. The number of distinct s-units in MIA.GRID.T10K dataset is also much larger, which is 4.47 times larger than that of SJ.GRID.T10K dataset and is 5.77 times larger than that of ATL.GRID.T10K dataset. The mean trajectory length and number of distinct s-units in ATL.GRID.T10K dataset are slightly smaller than those of SJ.GRID.T10K dataset. However, the number of trajectory patterns in ATL.GRID.T10K dataset is about 6 times larger than that of the other two datasets, given cell size = $[186m \times 220m]$ and $min\_sup = 0.008$. It results in TrajPod runs longer for ATL.GRID.T10K dataset than for SJ.GRID.T10K dataset.

4.5.4 Varying Data Sizes

In this set of experiments, we test the scalability and stability of TrajPod by varying the sizes of the input trajectory dataset with both grid-based and segment-based s-units. The value of min_sup is set to 0.006 and 0.008. The number of MO trajectories is ranging from 20K to 100K generated on SJ road network. The scalablity testing results are shown in Figure 40. The name of a data series displays the s-unit type of the dataset and the min_sup value set for the experiment. For example, $grid - 0.006$ data series shows the running times of TrajPod for grid-based trajectory datasets with $min\_sup = 0.006$. All resulting data series in Figure 40 show that TrajPod

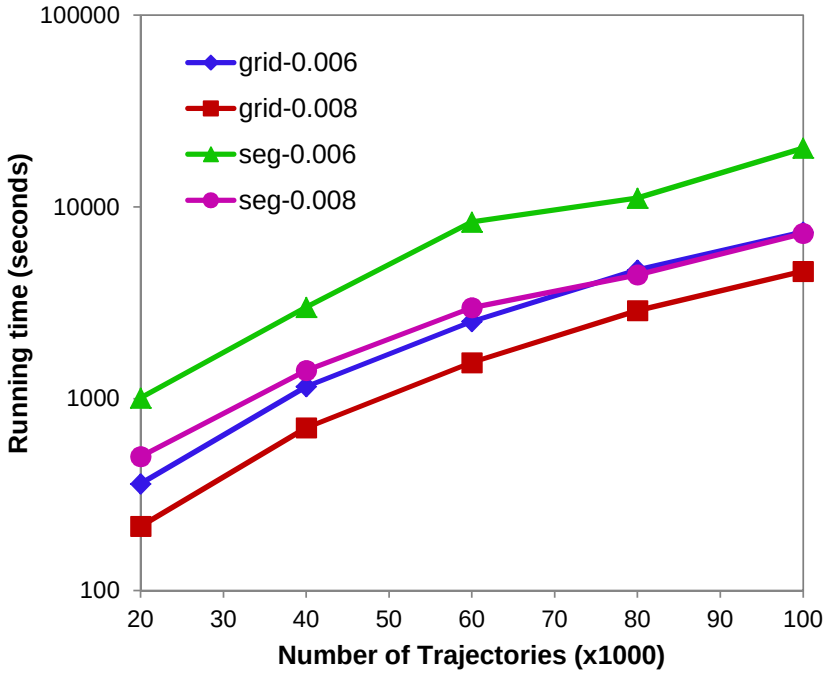
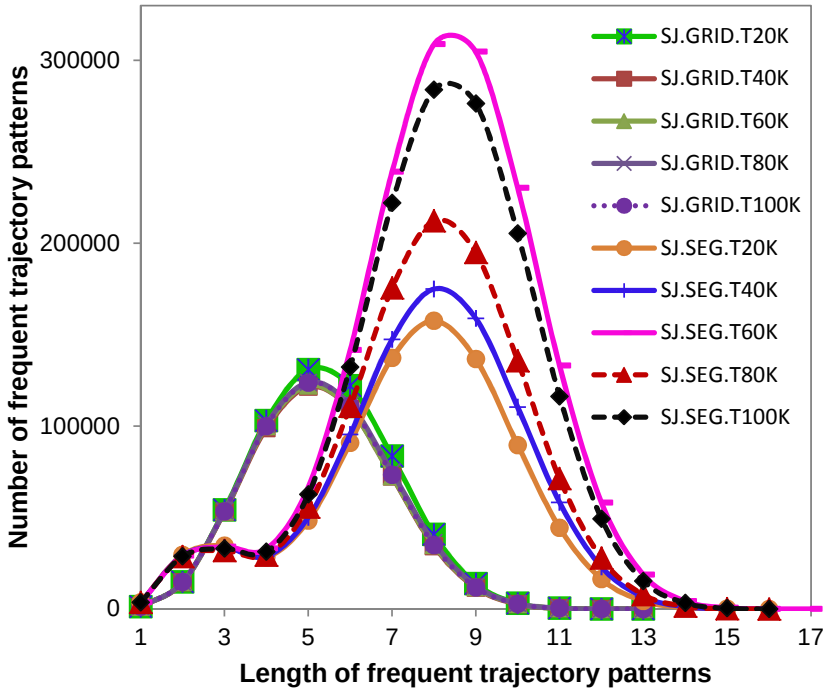


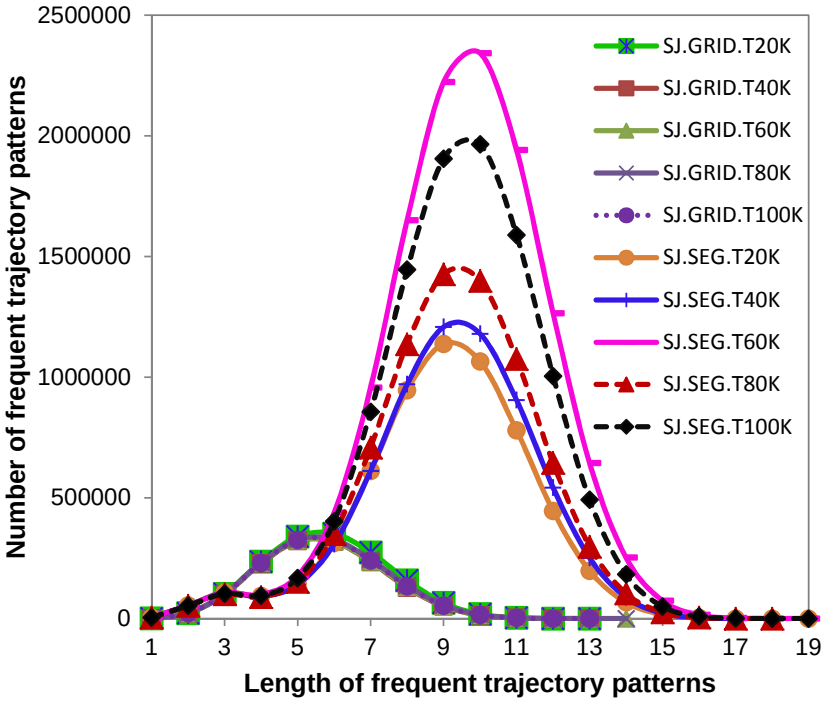
Figure 40: Running times of TRAJPOD vs. varying database sizes

is scalable to the varying dataset sizes. The running time is increased relatively smoothly as the dataset size grows. With the cell size of $[176m \times 218m]$ on SJ map, each grid cell contains an average of 3.56 road segments. Therefore, the s-unit cardinality of a dataset, the lengths of trajectories in terms of number of s-unit and the number of trajectory patterns in a dataset are smaller for GRID mode than for those of SEG mode. We can see the differences of the trajectory pattern distributions for grid-based datasets compared to segment-based datasets in Figure 41. Thus, for the same value of min_sup, TRAJPOD runs faster for grid-based trajectory datasets than segment-based trajectory datasets. Also, changes in the running time differences for different types of s-units, shown by the gaps between the data series of both s-unit types, are relatively similar.

In summary, our extensive experimental results have demonstrated the efficiency and effectiveness of the TRAJPOD algorithm. Overall, TRAJPOD outperforms other approaches and is very robust to different scales of maps and varying distribution



(a) $min\_sup = 0.008$



(b) $min\_sup = 0.006$

Figure 41: The distribution of frequent trajectory patterns of SJ.SEG datasets vs SJ.GRID datasets.

of frequent patterns within a trajectory dataset when varying the minimum support thresholds as well as different settings of grid cell sizes. TrajPod is very memory-stable and scalable to varying sizes of trajectory datasets.

4.6 Related Work

Many studies on finding interesting movement patterns in trajectory data have been proposed, for example, finding major traffic flow patterns [40], moving cluster patterns [50], maximal periodic patterns [19], periodic behaviors [60], longest duration flocks [37], leadership patterns [12], to name a few.

To the best of our knowledge, only a few related methods for sequential pattern mining of mobile object trajectories have been proposed. Since the granularity level of raw GPS locations is very low, an exact pattern of GPS locations hardly occurs multiple times in a trajectory dataset. Thus, the traditional notions of sequential patterns in transactional data cannot be applied directly for MO trajectory sequential pattern mining. Certain previous work [64] defines trajectory patterns as sequences of grid-based notations and mines them using the PrefixSpan algorithm. The work in [34] also adopts the PrefixSpan algorithm to mine trajectory patterns as sequences of regions of interests (ROIs) incorporate with the time interval between the ROIs. Approaches in [20] and [93] only mine patterns as sequences of consecutive elements within trajectories. Note that the standard sequential pattern is a sequence of elements which do not need to be consecutive. The work in [20] considers patterns as consecutive line segments and aims to find closed patterns. Other work [93] mines patterns as sequences of consecutive locations and searches for normalized matching (NM) between trajectories and patterns. In their approach, the Apriori property no longer holds for the resulting frequent patterns and a new min-max property of their defined NM measure is used instead. Our TrajPod approach generalizes the

trajectory pattern abstraction with any semantic spatial cover scheme for MO trajectories, such as grid-based and road segment-based abstraction and proposes novel technical solutions for the pattern mining process. TrajPod is the first work targeting trajectory pattern mining which aligns with the traditional sequential pattern mining research. We improve Apriori-based mining framework with space-efficient data structures and their corresponding locality-aware operators. Therefore, TrajPod is able to incorporate early and deep candidate pruning and fast support testing into the generation-and-test process specialized for mining MO trajectory patterns.

4.7 Conclusion

We have presented TrajPod, a fast and effective sequential pattern mining algorithm specialized for mobile object trajectories. TrajPod discovers the complete set of frequent trajectory patterns as sequences of semantic spatial units, such as grid cells or road segments. We introduce new data structures for representing MO trajectories, i.e., the locality-aware partitioned occurrence id-lists (pod-lists), which are the building blocks of the TrajPod algorithm. The use of pod-lists enables early and deep candidate pruning and fast support counting. Thus, TrajPod achieves both space efficiency and computational efficiency. A comprehensive performance study shows that TrajPod outperforms state-of-the-art sequential pattern mining algorithms by an order of magnitude. TrajPod is robust under different settings of road networks, minimum support thresholds, types of semantic spatial units and is scalable to increasing size of MO trajectory datasets.

CHAPTER V

TRAJBOX: TRAJECTORY PROCESSING AND MINING SOFTWARE FRAMEWORK

5.1 Overview

We design TrajBox, a framework that enables trajectory modeling and computation for querying and mining large collections of MO trajectories. TrajBox provides a programming interface to enable the development of MO trajectory-based applications. The API currently supports basic MO trajectory modeling and manipulating operations and supplies the MO trajectory clustering and pattern mining algorithms presented in this dissertation. In addition, TrajBox provides some related traditional clustering and sequential mining algorithms which are implemented particularly for both road network trajectories and free-space trajectories.

5.2 Design of TrajBox *System and API*

The TrajBox system design uses a 3-tier client/server architecture. Each client node acts as a mobile device which records its locations, sends its trajectories to a TrajBox server and makes requests to the server to get trajectory mining results for a particular road network. TrajBox server also distributes trajectory datasets across multiple nodes in a cluster. TrajBox is equipped with a suite of utility functions for trajectory data including preprocessing, indexing, querying, clustering and sequential pattern mining.

5.2.1 Trajectory Modeling

Road network-aware preprocessing. Input trajectory data is in the form of sequences of raw GPS points, i.e., geometric coordinates. When raw trajectories are

sent to the TrajBox server, they are map-matched [89] to the corresponding road network to form MO trajectories as sequences of road network locations. Figure 42 shows a sample of trajectory points consisting of Trajectory Id $Trid$, (x, y) coordinates ordered by timestamp and the corresponding map-matched segment Id *segid* of the road segment where the point resides.

Trid	SegId	X	Y
14	6596	737846.1	3739133.0
79	8780	736952.9	3739861.0
366	8777	736761.8	3739969.0
176	8776	736655.5	3740030.0
481	6600	738177.9	3738908.0
49	8446	733127.4	3744854.0
251	8785	736218.9	3740248.0
494	6597	737926.0	3739059.0
420	8783	737242.2	3739779.0
463	8776	736655.5	3740030.0
163	6554	738804.3	3738926.0
147	3207	733624.0	3743740.0
144	3209	733909.2	3743747.0
265	8779	736876.3	3739902.0
404	2055	731889.8	3746193.0
393	6646	736919.2	3738669.0
160	8786	736305.0	3740169.0
57	8775	736626.6	3740038.0
295	8775	736626.6	3740038.0
378	8791	737407.9	3739761.0
218	8314	732275.4	3745667.0
202	5972	737635.1	3739598.0
109	6611	738004.9	3738903.0
401	6593	738382.0	3738915.0
154	6000	737787.3	3739192.0
222	8788	736505.8	3740063.0
130	5985	737766.0	3739266.0
284	3287	734041.9	3743684.0
248	6605	738282.7	3738912.0
253	8444	733281.9	3744697.0
424	3771	734174.4	3743260.0

Figure 42: Sample trajectory data

Trajectory Abstraction. We provide several abstraction functions for trajectories in order to transform each original trajectory as a sequence of location points into a sequence of desired spatial units with respect to specific mining tasks including:

trajectory fragments (t-fragment), semantic spatial units (s-unit) and line segments. We denote a transformed trajectory $\mathcal{T}$-trajectory.

- t-fragment operator: This function splits a MO trajectory into a set of t-fragment, which is the subsequence of road network locations in the trajectory which share the same *segid*.

- s-unit operator: This function currently supports grid-based s-units and segment-based s-units. It takes as input an s-unit setting and a MO trajectory and its associated road network to transform the trajectory into a sequence of s-units.

- Line segment operator: This function performs polyline simplification to reduce the number of points of a trajectory while maintaining its basic geometric shape. It implements the popular Douglas-Peucker algorithm [30] which takes as input a trajectory and a tolerance distance threshold to transform the trajectory into sequence of line segments. Users can apply this function to abstract trajectories when not given a reference road network or free-space trajectories, such as the trajectories recording the movement of animals in a forest or hurricane trajectories.

5.2.2 Trajectory Manipulating

We provide a number of simple manipulating operations for MO trajectories and $\mathcal{T}$-trajectories including unit counting, extraction, intersection, k-subpattern.

- Unit counting: This operation returns the cardinality, i.e., the number of t-fragment, s-unit or line segments of a $\mathcal{T}$-trajectory.

- Spatio-extracting: This operation takes as input a trajectory and a segment ID *segid* or a grid cell ID *cellid* to return a t-fragment associated with *segid* or a subtrajectory covered by *cellid*.

- Temporal-extracting: Given a time window $[startTime, endTime]$ and a trajectory Tr, this function extracts the subtrajectory consisting of locations recorded in Tr during the given time window. This is very useful when users only want to analyze a subset of a trajectory dataset which records the movement in certain time intervals, such as during rush hours or in weekends.

- Intersection: This function computes the common parts in the form of a sequence of s-units from two trajectories.

- k-subpattern: Given a positive integer k, this function scans a $\mathcal{T}$-trajectory and outputs its complete set of k-subpatterns. It returns *null* when k is larger than the s-unit cardinality of the trajectory.

5.2.3 Trajectory Distance and Querying Functions

Distance functions. A distance function, which measures the distance/dissimilarity of two trajectories, is the centerpiece in trajectory clustering. We provide implementations of segment-based distance SegSD and grid-based distance functions SGBD, GridCSD which are presented in our TraceMob clustering framework (Chapter 3). Euclidean-based distances [16, 36] are also provided for free-space trajectories.

Trajectory querying. Given one of the distance functions above, a trajectory Tr and a distance threshold ϵ, a ϵ-querying function returns all trajectories in the dataset within the ϵ radius to Tr. A kNN-querying function compute the nearest k trajectories to Tr.

5.2.4 Operations for Trajectory-based Collections

By trajectory-based collections, we refer to the core concepts used in our NEAT algorithms (chapter 2) including base clusters (groups of t-fragment), flow clusters (groups of base clusters) and f-neigborhoods (groups of base clusters sharing one common road junction). TrajBox supplies the $netflow$ functions for two base clusters, two

flow clusters and the *merging selectivity* function to compute the relative traffic factor value of a base cluster with respect to its f-neigborhood as used in NEAT. These functions are very helpful when users want to analyze the traffic statistics in some particular road segments and junctions in a road network.

5.2.5 Trajectory Mining Algorithms

TrajBox provides the implementations of our trajectory mining algorithms for MO trajectories in road networks, including the road-network aware clustering algorithm NEAT, the road-network aware trajectory transformation algorithm $TrajMap$ used in our TraceMob framework and the frequent trajectory pattern mining TrajPod. In addition, it also includes a number of traditional mining algorithms such as Traclus [36], DBSCAN [31], PAM [53] for clustering, and SPADE [97], SPAM [15] and PrefixSpan [69] for sequential pattern mining specialized for trajectory data.

5.3 Sample Clustering Application using TrajBox *Framework*

By providing a programming interface, TrajBox is able to support the development of many trajectory-based applications. We built the NEAT clustering prototype system as a sample trajectory-based application system using TrajBox API to illustrate the usability of TrajBox framework.

The input for the NEAT system includes a trajectory dataset, a road network map corresponding to the trajectory dataset and a set of parameters. The trajectory dataset is provided by a trajectory database. The road network is supplied by a road network database. The set of parameters is either system-supplied or user-supplied. Figure 43 shows the architecture of our NEAT clustering system which is composed of an input preprocessor, a clustering engine and an output postprocessor. In addition, the NEAT GUI provides a user interface for users to interact with the system. Some screenshots of the NEAT GUI are shown in Figure 44, Figure 45 and Figure 46.

First, the trajectory dataset is preprocessed by the input preprocessor if necessary.

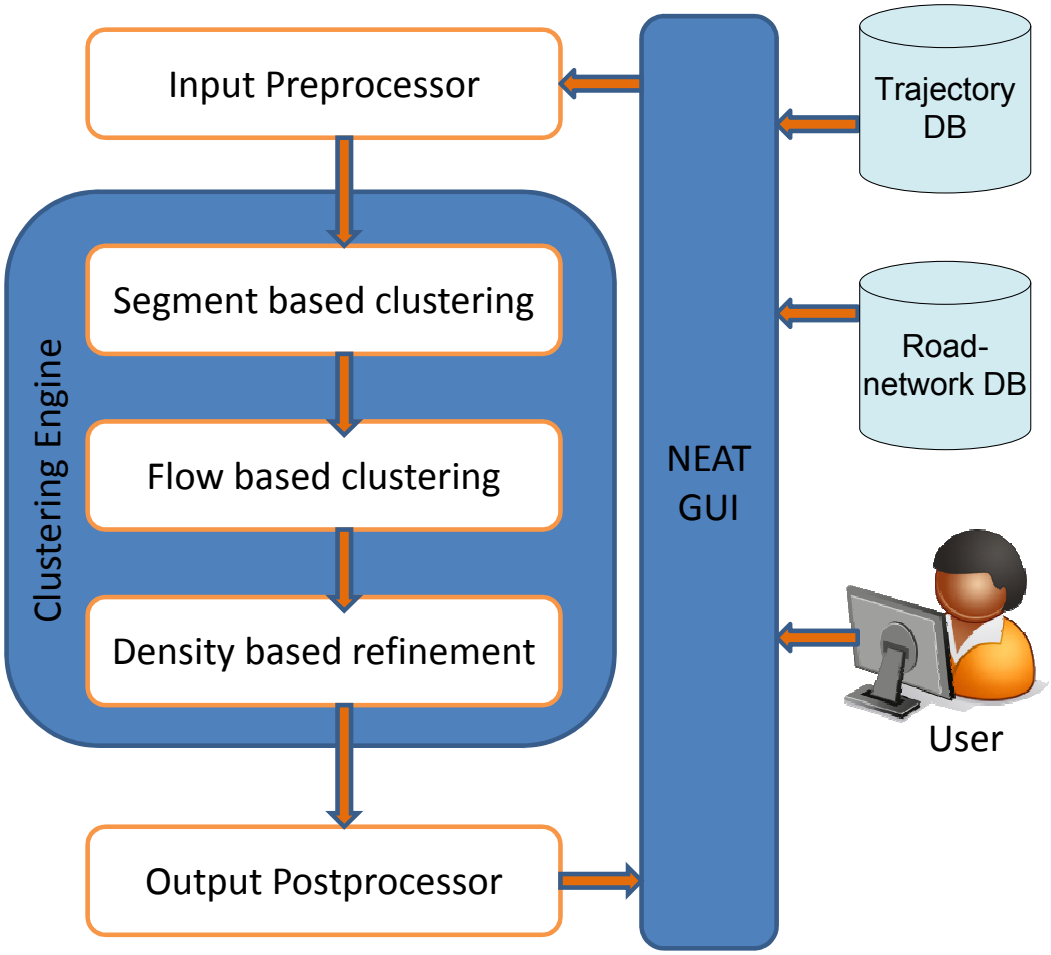


Figure 43: Neat system architecture

When a given set of trajectories are expressed in terms of a time series of geometric coordinates, NEAT will first preprocess the set of trajectories using the SLAMM map-matching algorithm [89] such that each point in a trajectory is mapped to a road network location which belongs to a specific road segment in the road network map. The input preprocessor produces a set of MO trajectories as sequences of road network locations. Next, the clustering engine runs the NEAT clustering algorithms on the given set of MO trajectories. Finally, the clustering result is visualized by the output postprocessor and returned to users. Figure 47 shows a screenshot visualizing three-phase clustering (opt-NEAT) results for a San Jose dataset.

Choose a dataset

ATLANTA Road Network

Choose a NEAT task

base-NEAT

Make it NEAT! (*)

A .jnlp file *neat_[timestamp].jnlp* will be generated. Save and open it to launch the application.

Map information:

Regions	Total length	# Segments	# Junctions	Avg. segment length	Junction degrees
North West Atlanta, GA	1384.4 km	9187	6979	150.7 m	avg: 2.6, max: 6

Dataset information:

Number of trajectories: 500
Number of points: 114,878

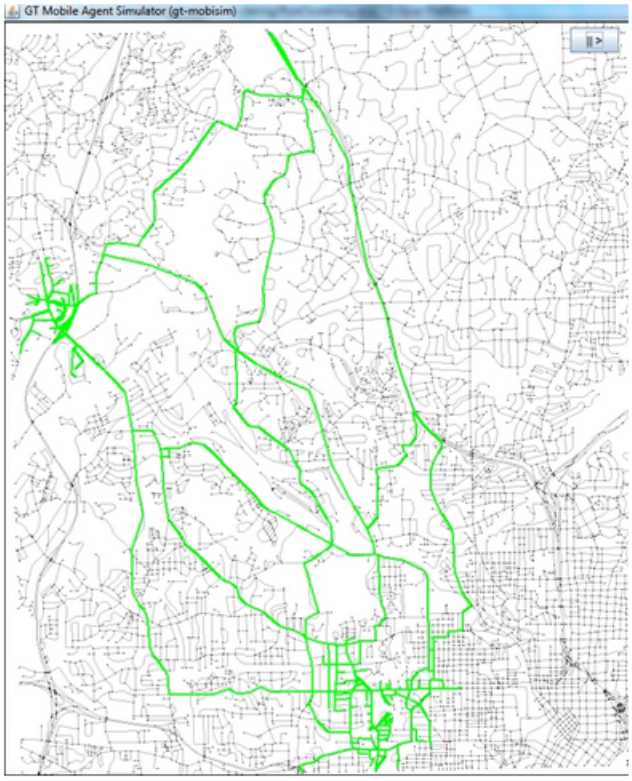


Figure 44: Neat GUI with a selected Atlanta dataset to run base-NEAT

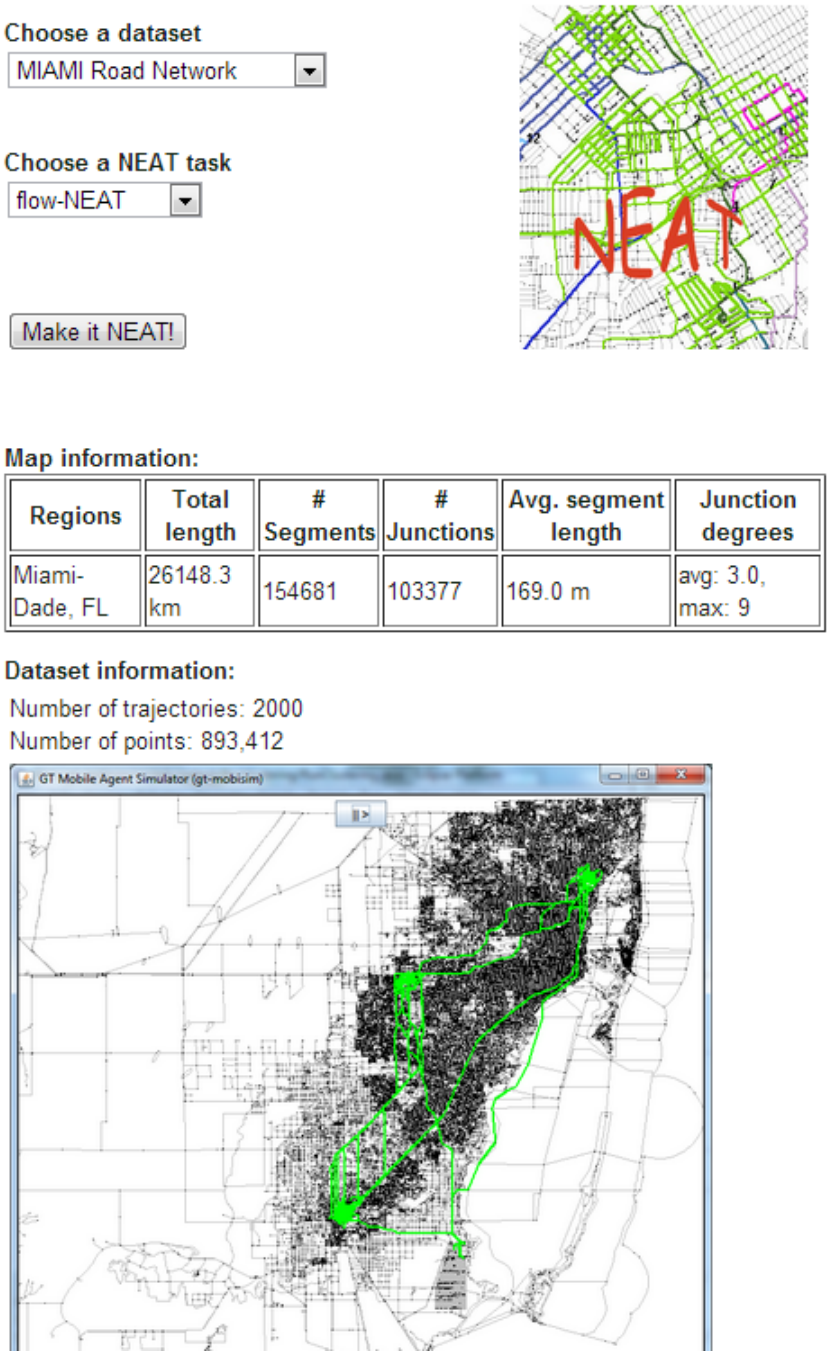


Figure 45: Neat GUI with a selected Miami-Dade dataset to run flow-NEAT

5.4 Conclusion

We have presented the design of TrajBox, a software programming framework for trajectory processing and mining with a sample MO trajectory clustering application

developed on top of TrajBox. The TrajBox system and APIs provide building blocks for trajectory analysis including a set of operations implementing the technical components presented in the previous chapters of this dissertation. Several software systems for traditional data mining [39] and free-space trajectories [11, 61] have been introduced. However, a software system for constrained MO trajectories is still in need. Our ultimate goal is to fully develop TrajBox as a software system that provides a rich API and a user-friendly interface specialized for analyzing and mining mobile object trajectories in road networks. We hope this software framework is a helpful toolkit to further studies in the growing field of trajectory mining and analysis.

Choose a dataset

SAN JOSE Road Network

Choose a NEAT task

opt-NEAT

Distance threshold (m)

2000.0

Make it NEAT!

Map information:

Regions	Total length	# Segments	# Junctions	Avg. segment length	Junction degrees
West San Jose, CA	1821.2 km	14600	10929	124.7 m	avg: 2.7, max: 6

Dataset information:

Number of trajectories: 1000
Number of points: 255,162

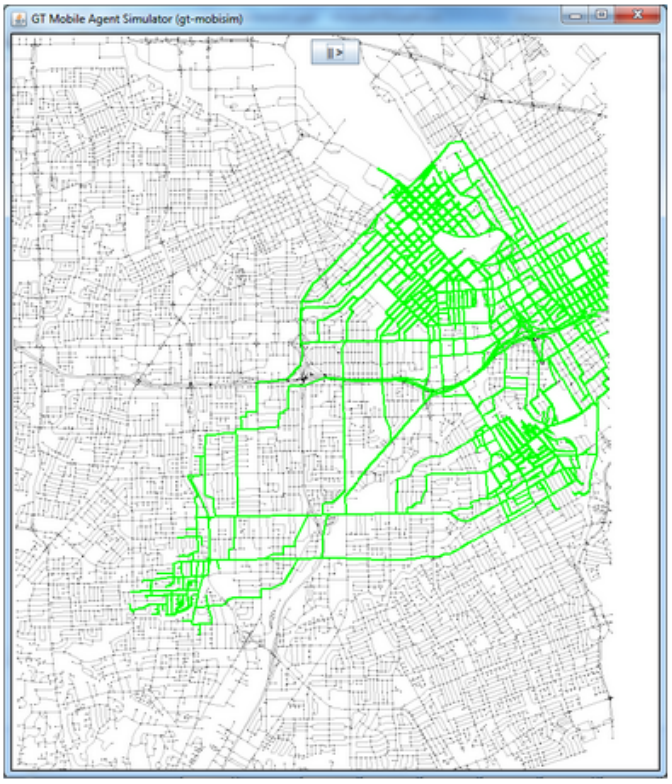


Figure 46: Neat GUI with a selected West San Jose dataset to run opt-NEAT

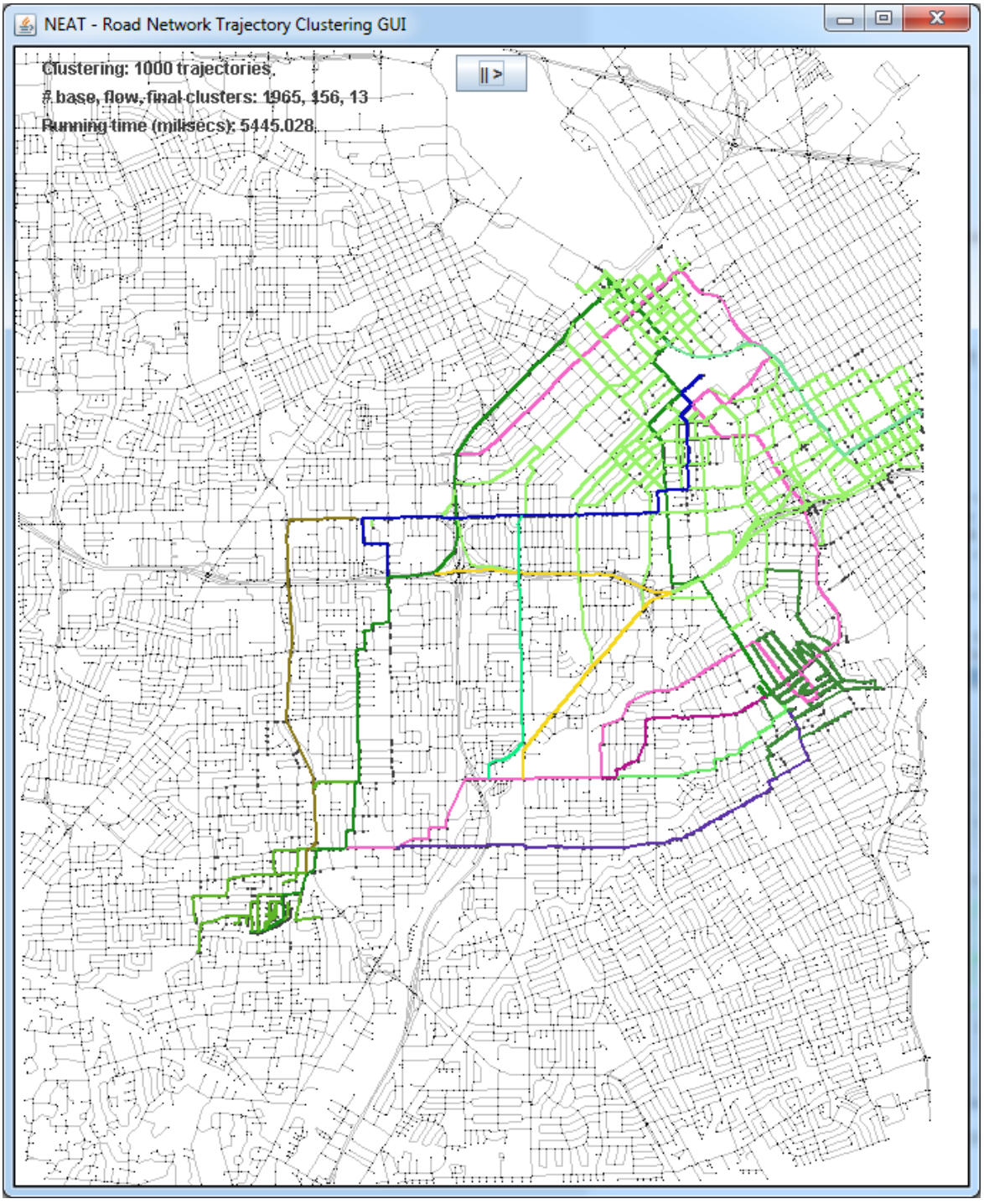


Figure 47: Neat output visualization

CHAPTER VI

CONCLUSION AND FUTURE WORK

This dissertation centers around processing, clustering and mining trajectories of mobile objects traveling in road networks, an emerging class of ubiquitous location-based data analysis. Through our studies, we have demonstrated that traditional mining algorithms, when applied directly to analyze MO trajectory data, suffer from poor performance and fail to deliver accurate and meaningful mining results. We carefully study the complex spatio-temporal characteristics of MO trajectories in the context of constrained road networks to design efficient and effective data representations and computational models for trajectory data to facilitate large-scale trajectory clustering and trajectory sequential pattern mining. In this chapter, we conclude our contributions and discuss future research directions.

6.1 Concluding Remarks

This dissertation covers the problem of trajectory clustering for both subtrajectories and whole trajectories. We have presented NEAT, a comprehensive road-network aware subtrajectory clustering framework that quickly discovers spatial clusters of MO subtrajectories representing major traffic flows in a road network. A salient feature of NEAT is its design of a configurable three-phase clustering framework in which each phase has its own goal in terms of delivering interesting trajectory clustering results to location-based applications. We have proposed TraceMob, a methodical framework for clustering whole trajectories. TraceMob provides an accurate and efficient distance measure between whole trajectories and a road-network aware trajectory transformation algorithm to effectively shift the clustering task for whole MO trajectories from the complex road network space into multidimensional

data clustering in an Euclidean space.

The dissertation also addresses the challenging sequential pattern mining problem for MO trajectory data, in which the conventional notions of frequent sequential patterns are usually not applicable. We have presented TrajPod, a fast locality-aware trajectory pattern mining algorithm which utilizes innovative data structures and computation schemes to discover the complete set of frequent semantic location sequences from large-scale MO trajectory datasets.

Finally, we introduce the design of the TrajBox software toolkit that provides basic MO trajectory modeling and manipulating operations and supplies a suite of novel MO trajectory clustering and pattern mining algorithms presented in this dissertation.

6.2 Future Directions

We have contributed technical solutions for critical problems in mining mobile object trajectories. Our trajectory mining frameworks and algorithms are designed in a configurable and extensible fashion, which enables a number of interesting extensions and future research directions.

6.2.1 MO Trajectory Clustering

The purpose of clustering whole trajectories is to discover the grouping structure in a given trajectory dataset. One may raise a question whether the grouping structure of MO trajectories in a specific road network is stable or dynamically changing over time. One observation is that the daily/weekly commutes of a person usually remain unchanged. In addition, today people mostly use GPS-based devices or map applications for navigating and often follow shortest routes to travel within a road network, resulting in a high level of overlapping in MO trajectories. Because of that, do we really need to store every single trajectory and cluster all historical trajectory data which we have to deal with unnecessary big data workloads? Or can we execute some trajectory profiling and updating strategies that can store MO trajectories in a

compact and representative data model without the loss of interesting changes that may happen in space and time with regard to user movement behaviors? These are interesting open issues which need to be addressed to facilitate trajectory clustering in both batch processing and real-time processing scenarios. The study of MO trajectory clustering structure dynamics in which we need to detect and analyze significant changes in the clustering structure of MO trajectories, for example in the presence of extraordinary events like disasters or wars, associated with a road network over time is also a promising mining problem.

Another critical problem in MO trajectory clustering is that we need a set of standard metrics for trajectory cluster quality, which greatly helps in evaluating the correctness of trajectory clustering algorithms. So far, no good cluster validation measurement for MO trajectory clusters has been proposed. In TraceMob, we validate clustering results in two spaces: the multidimentional space where $k$means clustering is performed, and the original road network space where clusters of multidimensional points are mapped back to clusters of trajectories. In the former space, we use a traditional cluster validation measure. In the latter space, we adapt another popular cluster validation measure for the trajectory clusters. Although our proposed cluster validation measurement can reflect the comparable qualities of different clustering results in the road network space, the quality differences are not as clear as measured in a metric space for clusters of multidimentional data points.

6.2.2 Mining interesting movement patterns in MO trajectories

Our NEAT approach provides a clustering-based perspective to find interesting movement and traffic patterns hidden in MO trajectory data. In the development of NEAT, we consider road segment-based subtrajectory, i.e., a t-fragment, as the default unit for clustering. We can integrate a user-defined fragment into our framework to let users define the granularity of fragments. For examples, a user can define the grid cells

covering a road network as the building blocks in the base cluster formation phase. Then the same merging and refinement process can be applied to the resulting cell-based or any user-defined fragment clusters. In addition, in the last clustering phase, beside our modified Hausdorff distance, other distance functions that introduce interesting optimization results can also be included in this phase. The endpoints of the flow clusters can be considered as places of interest. Thus, we can extend our modified DBSCAN-like optimization phase to group the set of endpoints of the resulting flow clusters to discover regions of interest inherent in the given trajectory data. Furthermore, consider that each road network location is recorded with a specific timestamp, another interesting extension to NEAT is to consider the temporal information contained in the MO trajectories. We can apply NEAT to an extracted sub-trajectory set from the given trajectories, e.g., recorded mobility traces in rush hours or during weekends, to discover the traffic patterns during different time windows (in terms of hour, day, week, month or year).

Parallelizing the NEAT algorithms is in our future development plan. Specifically, in the first phase, the same method of splitting trajectories and putting trajectory fragments into their associated base clusters can run for each trajectory in parallel. In the second phase, we would divide the map into partitions and perform MapReduce-like jobs to retrieve the flow clusters. The last phase is essentially traditional clustering which can adapt many available parallelized versions of traditional clustering algorithms such as ones in [5].

6.2.3 Mobile Object Trajectory Pattern Mining

Much research is needed in distributed trajectory pattern mining to achieve high performance and high scalability in mining very large trajectory datasets. TrajPod, in particular, can be parallelized by locality-aware partitioning the vertical layout of the datasets, i.e., using a one-to-one mapping from partitions of the road network to

partitions of is associated semantic spatial coverage. First, TrajPod will execute in parallel on each partition. Next, there needs to have a hierarchical level-wise merging process for the immediate frequent trajectory pattern trees to retrieve the final set of frequent trajectory patterns. In general, we need to consider both vertical and horizontal partitioning of the trajectory dataset depending on what data structure we use in the pattern mining process. In addition, a fault tolerance scheme specialized for distributed trajectory pattern mining is beneficial in the presence of system failure. Integrating partial materialization and support counting operator tracking, similar to the approach in [44] is one possibility for fault tolerance, so that we do not have to restart the mining process from scratch. Moreover, we also need a new set of domain specific criteria to rank the discovered frequent trajectory patterns to achieve the most desired MO trajectory patterns.

REFERENCES

[1] http://www.abiresearch.com/.

[2] http://www.strategyanalytics.com/.

[3] http://www.gartner.com/.

[4] http://www.cc.gatech.edu/projects/disl/.

[5] http://www.mahout.apache.org/.

[6] "GeoLife GPS trajectories." http://research.microsoft.com/en-us/projects/geolife/.

[7] "R-tree Portal." http://rtreeportal.org/.

[8] Aggarwal, C. C., Han, J., Wang, J., and Yu, P. S., "A framework for clustering evolving data streams," in *Proceedings of the 29th international conference on Very large data bases - Volume 29*, VLDB '03, pp. 81–92, VLDB Endowment, 2003.

[9] Agrawal, R. and Srikant, R., "Fast algorithms for mining association rules in large databases," in *Proceedings of the 20th International Conference on Very Large Data Bases*, VLDB '94, (San Francisco, CA, USA), pp. 487–499, Morgan Kaufmann Publishers Inc., 1994.

[10] Agrawal, R. and Srikant, R., "Mining sequential patterns," in *Proceedings of the Eleventh International Conference on Data Engineering*, ICDE '95, (Washington, DC, USA), pp. 3–14, IEEE Computer Society, 1995.

[11] Ananthanarayanan, G., Haridasan, M., Mohomed, I., Terry, D., and Thekkath, C. A., "Startrack: a framework for enabling track-based applications," in *Proceedings of the 7th international conference on Mobile systems, applications, and services*, MobiSys '09, (New York, NY, USA), pp. 207–220, ACM, 2009.

[12] Andersson, M., Gudmundsson, J., Laube, P., and Wolle, T., "Reporting leaders and followers among trajectories of moving point objects," *GeoInformatica*, vol. 12, no. 4, pp. 497–528, 2008.

[13] Ankerst, M., Breunig, M. M., Kriegel, H.-P., and Sander, J., "Optics: ordering points to identify the clustering structure," in *Proc. SIGMOD'99*, pp. 49–60, 1999.

[14] Arthur, D. and Vassilvitskii, S., "k-means++: the advantages of careful seeding," in *in Proc. SODA '07*.

[15] Ayres, J., Flannick, J., Gehrke, J., and Yiu, T., "Sequential pattern mining using a bitmap representation," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, (New York, NY, USA), pp. 429–435, ACM, 2002.

[16] Black, P. E., "Hausdorff distance," Dec. 2004. in Dictionary of Algorithms and Data Structures.

[17] Black, P. E., "Manhattan distance," May 2006. in Dictionary of Algorithms and Data Structures.

[18] Brakatsoulas, S., Pfoser, D., and Tryfona, N., "Practical data management techniques for vehicle tracking data," in *Data Engineering, 2005. ICDE 2005. Proceedings. 21st International Conference on*, pp. 324–325, 2005.

[19] Cao, H., Mamoulis, N., and Cheung, D. W., "Discovery of periodic patterns in spatiotemporal sequences," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 4, pp. 453–467, 2007.

[20] Cao, H., Mamoulis, N., and Cheung, D., "Mining frequent spatio-temporal sequential patterns," in *Data Mining, Fifth IEEE International Conference on*, pp. 8 pp.–, 2005.

[21] Cao, X., Cong, G., and Jensen, C. S., "Mining significant semantic locations from gps data," *Proc. VLDB Endow.*, vol. 3, Sept. 2010.

[22] Chakka, V. P., Prasad, V., Adam, C., Everspaugh, A. C., and Patel, J. M., "Indexing large trajectory data sets with SETI," in *Proc. CIDR'03*.

[23] Chatterjee, M., Das, S., and Turgut, D., "Wca: A weighted clustering algorithm for mobile ad hoc networks," *Cluster Computing*, vol. 5, no. 2, pp. 193–204, 2002.

[24] Chen, L. and Ng, R., "On the marriage of lp-norms and edit distance," in *Proc. VLDB '04*.

[25] Chen, L., Özsu, M. T., and Oria, V., "Robust and fast similarity search for moving object trajectories," in *Proc. SIGMOD '05*.

[26] Chen, Y. and Tu, L., "Density-based clustering for real-time stream data," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, (New York, NY, USA), pp. 133–142, ACM, 2007.

[27] Chon, H. D., Agrawal, D., and El Abbadi, A., "Range and kNN query processing for moving objects in grid model," *Mob. Netw. Appl.*, vol. 8, Aug. 2003.

[28] De Almeida, V. T. and Güting, R. H., "Indexing the trajectories of moving objects in networks*," *Geoinformatica*, vol. 9, pp. 33–60, Mar. 2005.

[29] Dijkstra, E. W., "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959.

[30] Douglas, D. and Peucker, T., "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, Dec. 1973.

[31] Ester, M., Kriegel, H.-P., Sander, J., and Xu, X., "A density-based algorithm for discovering clusters in large spatial databases with noise," in *Proc. SIGKDD'96*, pp. 226–231, 1996.

[32] Faloutsos, Christos and Lin, King-Ip, "Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets," in *Proc. SIGMOD'95*.

[33] Gaffney, S. and Smyth, P., "Trajectory clustering with mixtures of regression models," in *Proc. SIGKDD'99*, pp. 63–72, 1999.

[34] Giannotti, F., Nanni, M., Pinelli, F., and Pedreschi, D., "Trajectory pattern mining," in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '07, (New York, NY, USA), pp. 330–339, ACM, 2007.

[35] Gidofalvi, G., Larsen, H. R., and Pedersen, T. B., "Estimating the capacity of the location - based advertising channel," *Int. J. Mob. Commun.*, vol. 6, pp. 357–375, March 2008.

[36] gil Lee, J., Han, J., and young Whang, K., "Trajectory clustering: a partition-and-group framework," in *Proc. SIGMOD'07*, pp. 593–604, 2007.

[37] Gudmundsson, J. and van Kreveld, M., "Computing longest duration flocks in trajectory data," in *Proceedings of the 14th annual ACM international symposium on Advances in geographic information systems*, GIS '06, (New York, NY, USA), pp. 35–42, ACM, 2006.

[38] Guha, S., Rastogi, R., and Shim, K., "Cure: an efficient clustering algorithm for large databases," in *Proc. SIGMOD'98*, pp. 73–84, 1998.

[39] Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H., "The weka data mining software: an update," *SIGKDD Explor. Newsl.*, vol. 11, pp. 10–18, Nov. 2009.

[40] Han, B., Liu, L., and Omiecinski, E., "Neat: Road network aware trajectory clustering," in *Proceedings of the 2012 IEEE 32nd International Conference on Distributed Computing Systems*, ICDCS '12, (Washington, DC, USA), pp. 142–151, IEEE Computer Society, 2012.

[41] Han, B., Liu, L., and Omiecinski, E., "Fast locality-aware trajectory pattern mining," tech. rep., 2013.

[42] Han, B., Liu, L., and Omiecinski, E., "A methodical approach to clustering whole trajectories of mobile objects in road networks," tech. rep., 2013.

[43] Han, B., Liu, L., and Omiecinski, E. R., "Road-network aware trajectory clustering: Integrating locality, flow and density," *IEEE Transactions on Mobile Computing*, vol. 99, no. PrePrints, p. 1, 2013.

[44] Han, B., Omiecinski, E., Mark, L., and Liu, L., "Otpm: Failure handling in data-intensive analytical processing," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*, pp. 35–44, 2011.

[45] Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., and Hsu, M.-C., "Freespan: frequent pattern-projected sequential pattern mining," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '00, (New York, NY, USA), pp. 355–359, ACM, 2000.

[46] Haridasan, M., Mohomed, I., Terry, D., Thekkath, C. A., and Zhang, L., "Startrack next generation: a scalable infrastructure for track-based applications," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, (Berkeley, CA, USA), pp. 1–6, USENIX Association, 2010.

[47] Hull, B., Bychkovsky, V., Zhang, Y., Chen, K., Goraczko, M., Miu, A., Shih, E., Balakrishnan, H., and Madden, S., "Cartel: a distributed mobile sensor computing system," in *Proceedings of the 4th international conference on Embedded networked sensor systems*, SenSys '06, (New York, NY, USA), pp. 125–138, ACM, 2006.

[48] Huttenlocher, D. P., Klanderman, G. A., Kl, G. A., and Rucklidge, W. J., "Comparing images using the Hausdorff distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, pp. 850–863, 1993.

[49] Jeong, J., Guo, S., Gu, Y., He, T., and Du, D. H. C., "TSF: Trajectory-based statistical forwarding for infrastructure-to-vehicle data delivery in vehicular networks.," in *Proc. ICDCS'10*, pp. 557–566, 2010.

[50] Kalnis, P., Mamoulis, N., and Bakiras, S., "On discovering moving clusters in spatio-temporal data," in *Advances in Spatial and Temporal Databases* (Bauzer Medeiros, C., Egenhofer, M., and Bertino, E., eds.), vol. 3633 of *Lecture Notes in Computer Science*, pp. 364–381, Springer Berlin Heidelberg, 2005.

[51] Kamar, E. and Horvitz, E., "Collaboration and shared plans in the open world: studies of ridesharing," in *Proc. IJCAI'09*, pp. 187–194, 2009.

[52] Kaufman, L. and Rousseeuw, P., *Finding groups in data: An introduction to cluster analysis*. 1990.

[53] Kaufmann, L. and Rousseeuw, P. J., "Clustering by means of medoids," *Statistical Data Analysis Based on the L_1-Norm and Related Methods*, 1987.

[54] Kendall, M. and Gibbons, J. D., *Rank Correlation Methods*. Edward Arnold, 1990.

[55] Keogh, E., "Exact indexing of dynamic time warping," in *Proc. VLDB '02*.

[56] Kharrat, A., Popa, I. S., Zeitouni, K., and Faiz, S., "Clustering algorithm for network constraint trajectories.," in *Proc. SDH'08*, pp. 631–647, 2008.

[57] Kim, S.-W., Park, S., and Chu, W. W., "An index-based approach for similarity search supporting time warping in large sequence databases," in *Proc. ICDE '01*.

[58] Lee, J.-G., Han, J., Li, X., and Gonzalez, H., "Traclass: Trajectory classification using hierarchical region-based and trajectory-based clustering,"

[59] Li, Z., Lee, J., Li, X., and Han, J., "Incremental clustering for trajectories," in *Proc. DASFAA'10*, pp. 32–46, 2010.

[60] Li, Z., Ding, B., Han, J., Kays, R., and Nye, P., "Mining periodic behaviors for moving objects," in *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '10, (New York, NY, USA), pp. 1099–1108, ACM, 2010.

[61] Li, Z., Han, J., Ji, M., Tang, L.-A., Yu, Y., Ding, B., Lee, J.-G., and Kays, R., "Movemine: Mining moving object data for discovery of animal movement patterns," *ACM Trans. Intell. Syst. Technol.*, vol. 2, pp. 37:1–37:32, July 2011.

[62] Lloyd, S. P., "Least squares quantization in PCM," *IEEE Transactions on Information Theory*, vol. 28, pp. 129–137, 1982.

[63] Mohan, P., Padmanabhan, V. N., and Ramjee, R., "Nericell: rich monitoring of road and traffic conditions using mobile smartphones," in *Proceedings of the 6th ACM conference on Embedded network sensor systems*, SenSys '08, (New York, NY, USA), pp. 323–336, ACM, 2008.

[64] Morzy, M., "Mining frequent trajectories of moving objects for location prediction," in *Machine Learning and Data Mining in Pattern Recognition* (Perner, P., ed.), vol. 4571 of *Lecture Notes in Computer Science*, pp. 667–680, Springer Berlin Heidelberg, 2007.

[65] Mouratidis, K., Papadias, D., and Hadjieleftheriou, M., "Conceptual partitioning: an efficient method for continuous nearest neighbor monitoring," in *Proc. SIGMOD '05*.

[66] Nanni, M. and Pedreschi, D., "Time-focused clustering of trajectories of moving objects," *J. Intell. Inf. Syst.*, vol. 27, pp. 267–289, Nov. 2006.

[67] Ng, R. T. and Han, J., "Clarans: A method for clustering objects for spatial data mining," *IEEE Trans. on Knowl. and Data Eng.*, vol. 14, pp. 1003–1016, Sept. 2002.

[68] Nievergelt, J., Hinterberger, H., and Sevcik, K. C., "The grid file: An adaptable, symmetric multikey file structure," *ACM Trans. Database Syst.*, vol. 9, 1984.

[69] Pei, J., Han, J., Member, S., Mortazavi-asl, B., Wang, J., Pinto, H., Chen, Q., Dayal, U., Society, I. C., Society, I. C., and chun Hsu, M., "Mining sequential patterns by pattern-growth: The prefixspan approach," *IEEE Transactions on Knowledge and Data Engineering*, vol. 16, p. 2004, 2004.

[70] Pesti, P., Bamba, B., Doo, M., Liu, L., Palanisamy, B., and Weber, M., "GTMobiSIM: A mobile trace generator for road networks." `http://code.google.com/p/gt-mobisim/`, Sept. 2009.

[71] Pesti, P., Liu, L., Bamba, B., Iyengar, A., and Weber, M., "Roadtrack: scaling location updates for mobile clients on road networks with query awareness," *Proc. VLDB Endow.*, vol. 3, pp. 1493–1504, Sept. 2010.

[72] Qin, H., Li, Z., Wang, Y., Lu, X., Zhang, W., and Wang, G., "An integrated network of roadside sensors and vehicles for driving safety: Concept, design and experiments," in *Pervasive Computing and Communications (PerCom), 2010 IEEE International Conference on*, pp. 79–87, 2010.

[73] Rodrigues, P. P., Gama, J. a., and Lopes, L., "Clustering distributed sensor data streams," in *Proceedings of the European conference on Machine Learning and Knowledge Discovery in Databases - Part II*, ECML PKDD '08, (Berlin, Heidelberg), pp. 282–297, Springer-Verlag, 2008.

[74] Roh, G.-P. and won Hwang, S., "Nncluster: An efficient clustering algorithm for road network trajectories.," in *Proc. DASFAA'10*, pp. 47–61, 2010.

[75] Rote, G., "Computing the minimum hausdorff distance between two point sets on a line under translation," *Information Processing Letters*, vol. 38, pp. 123–127, 1991.

[76] Rousseeuw, P., "Silhouettes: a graphical aid to the interpretation and validation of cluster analysis," *J. Comput. Appl. Math.*, vol. 20, Nov. 1987.

[77] Shekhar, S. and Xiong, H., *Encyclopedia of GIS*. Springer Publishing Company, Incorporated, 1st ed., 2007.

[78] SPADE code. http://www.cs.rpi.edu/~zaki/www-new/pmwiki.php/Software/Software.

[79] SPAM code. http://himalaya-tools.sourceforge.net/Spam/.

[80] Srikant, R. and Agrawal, R., "Mining sequential patterns: Generalizations and performance improvements," in *Proceedings of the 5th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '96, (London, UK, UK), pp. 3–17, Springer-Verlag, 1996.

[81] T. H. Cormen et al., *Introduction to Algorithms*. New York: The MIT Press, 2001.

[82] The ELBA project. http://www.e-lba.com.

[83] TIGER/Line. http://www.census.gov/geo/www/tiger.

[84] Treiber, M. and Kesting, A., *Traffic Flow Dynamics: Data, Models and Simulation*. Springer, 2012.

[85] USGS data (.svg). http://edc2.usgs.gov/geodata/index.php.

[86] Vlachos, M., Gunopoulos, D., and Kollios, G., "Discovering similar multidimensional trajectories," in *Proc. ICDE '02*.

[87] Šidlauskas, D., Šaltenis, S., Christiansen, C. W., Johansen, J. M., and Šaulys, D., "Trees or grids?: indexing moving objects in main memory," in *Proc. GIS '09*.

[88] Wang, Y.-X. and Bao, F., "An entropy-based weighted clustering algorithm and its optimization for ad hoc networks," in *Wireless and Mobile Computing, Networking and Communications, 2007. WiMOB 2007. Third IEEE International Conference on*, pp. 56–56, 2007.

[89] Weber, M., Liu, L., Jones, K., Covington, M. J., Nachman, L., and Pesti, P., "On map matching of wireless positioning data: a selective look-ahead approach," in *Proc. GIS'10*, pp. 290–299, 2010.

[90] Wei, D. and Chan, H., "Clustering ad hoc networks: Schemes and classifications," vol. 3, pp. 920–926, 2006.

[91] Won, J.-I., Kim, S.-W., Baek, J.-H., and Lee, J., "Trajectory clustering in road network environment," in *Computational Intelligence and Data Mining, 2009. CIDM '09. IEEE Symposium on*, pp. 299–305, 2009.

[92] Xu, H., Oh, L.-B., and Teo, H.-H., "Perceived effectiveness of text vs. multimedia location-based advertising messaging," *Int. J. Mob. Commun.*, vol. 7, pp. 154–177, January 2009.

[93] Yang, J. and Hu, M., "Trajpattern: Mining sequential patterns from imprecise trajectories of mobile objects," in *Advances in Database Technology - EDBT 2006* (Ioannidis, Y., Scholl, M., Schmidt, J., Matthes, F., Hatzopoulos, M., Boehm, K., Kemper, A., Grust, T., and Boehm, C., eds.), vol. 3896 of *Lecture Notes in Computer Science*, pp. 664–681, Springer Berlin Heidelberg, 2006.

[94] Yi, B.-K., Jagadish, H. V., and Faloutsos, C., "Efficient retrieval of similar time sequences under time warping," in *Proc. ICDE '98*.

[95] Yiu, M. L. and Mamoulis, N., "Clustering objects on a spatial network," in *Proc. SIGMOD'04*, pp. 443–454, 2004.

[96] Yu, X., Pu, K. Q., and Koudas, N., "Monitoring k-nearest neighbor queries over moving objects," in *Proc. ICDE '05*.

[97] Zaki, M. J., "Spade: An efficient algorithm for mining frequent sequences," in *Machine Learning*, pp. 31–60, 2001.

[98] Zhang, T., Ramakrishnan, R., and Livny, M., "Birch: an efficient data clustering method for very large databases," in *Proc. SIGMOD'96*, pp. 103–114, 1996.

[99] Zheng, Y., Zhang, L., Xie, X., and Ma, W.-Y., "Mining correlation between locations using human location history," in *Proc. GIS '09*, 2009.

VITA

Binh Han was born and raised in Bai Bang, a beautiful industrial town in the north of Vietnam. She received the B.Sc degree in Computer Science from Hanoi University of Science and Technology (HUST), Hanoi, Vietnam. During her undergraduate years, she worked as an Software Engineering intern at FPT and as a research intern at HUST's High Performance Computing Center. She worked as an IT specialist at Viettel Telecom after graduating from HUST. Binh came to Georgia Tech in 2009 to pursue the Ph.D degree in Computer Science, advised by Prof. Ling Liu. At Georgia Tech, she was affiliated with the Distributed Data-intensive Systems Lab (DiSL) and the Center for Experimental Research in Computer Systems (CERCS). Her dissertation research contributed novel mining algorithms and frameworks in location-based data mining. Her internships with PNNL and Amazon covered areas in scientific workflow systems, stream computing, big data analytics and business intelligence. Her list of honors includes the scholarships from the Vietnam Ministry of Education and Training, third prize in the 24th Student Scientific Research Contest at HUST and the VEF Fellowship for graduate study.