

國立交通大學

資訊科學與工程研究所

博士論文

軌跡模式探勘與應用之研究

A Study on Trajectory Pattern Mining and Applications

研究生：洪智傑

指導教授：彭文志 教授

1896

中華民國一〇一年七月

軌跡模式探勘與應用之研究
A Study on Trajectory Pattern Mining and Applications

研究生：洪智傑

Student : Chih-Chieh Hung

指導教授：彭文志

Advisor : Wen-Chih Peng

國立交通大學
資訊科學與工程研究所
博士論文

A Dissertation
Submitted to Department of Computer Science
College of Computer Science
National Chiao Tung University
in partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy
in
1896
Computer Science

July 2011

Hsinchu, Taiwan, Republic of China

中華民國一百年七月

學生：洪智傑

指導教授：彭文志

國立交通大學資訊科學與工程學系博士班

摘要

隨著行動裝置的普及，我們可以由多種的設備及來源收集使用者的軌跡資料。由於這些軌跡資料中含有使用者的移動資訊，因此由這些資料中可以探勘使用者的移動行為，此即為本篇論文中所提及之軌跡模式。這些模式在開發新的應用或是增加既有的位置感知應用上都極具價值。在本篇論文中，我們提出了一連串有關於軌跡模式探勘相關之演算法。我們首先提出如何大規模地收集使用者的軌跡資料。緊接著，我們提出兩個探勘軌跡模式的演算法。最後，我們利用使用者的軌跡模式來判斷使用者間的隱性社群關係。

在本篇論文的第一個主題，我們研究如何在由車輛所構成之感測網路中進行大規模收集軌跡資料的方式。我們提出了一個架構 MDC 來降低資料傳輸量以及車輛的回報數目。在 MDC 中，車輛利用模型來代表其感測到之讀數，並且利用匯集之方式使得有相類似模型的車輛只需回傳一份至伺服器，因此可以在收集軌跡資料時的資料傳輸量。

在本篇論文的第二個主題中，我們利用迴歸的方式自通聯記錄中探勘使用者的軌跡模式。由於通聯記錄只能夠反應使用者部份的行為，因此在本主題中，我們主要利用迴歸的方式自通聯記錄中復原使用者的軌跡模式。首先，我們先粹取出具有相同移動行為的通聯記錄，並將具有時間空間相關性的通聯記錄群聚起來。最後，使用者的移動模式便可以用數條迴歸曲線表示。

在本篇論文的第三個主題中，我們提出了利用軌跡中的線索(Clue) 探勘軌跡模式的演算法。在實際的軌跡中，有許多因素會造成其無法完整表達使用者的移動路徑。但是，縱使這些軌跡只能表達使用者部份的移動行為，但是我們依然可利用這些軌跡所隱含的資訊來進行使用者移動行為之探勘。因此，我們首先提出一利用兩軌跡中所含線索的多寡來衡量兩個軌跡相似度的方式，並利用這個相似度將具有相同移動行為的軌跡群聚起來，最後再將每一群中的軌跡之時間空間資訊匯集，推導出使用者的移動模式。

最後，我們利用使用者的軌跡模式來判斷使用者的隱性社群關係。我們可以觀察出，有相同移動行為的使用者極有可能在位置感知服務上或是現實生活上有所互動。因此，在這個主題中我們將判斷並比較使用者的軌跡模式，並將相同軌跡模式的使用者視為是一社群。首先，我們先將使用者的軌跡模式建成一個樹狀結構。利用 Editing distance 的概念，我們設計出比較使用者軌跡模式的距離函數。利用此距離函數，我們便可以推得出使用者的社群關係。



A Study on Trajectory Pattern Mining and Applications

Student: Chih-Chieh Hung

Advisor: Prof. Wen-Chih Peng

Department of Computer Science

National Chiao Tung University

ABSTRACT

With the pervasiveness of mobile devices, the location of users can be easily determined by either GPS devices or some positioning techniques. Moreover, wireless communication systems enable users to access various kinds of information from anywhere at any time. Nowadays, through smart phones or some portable devices, people could access location-based services or share their locations to their friends via social web sites, such as Google's latitude service and Foursquare service. These phenomena show that there will be an increasing amount of user trajectory data available. It is a challenge and interesting task to discover valuable knowledge. With knowledge mined, we could develop many novel applications from such a huge amount of user trajectory data.

In this dissertation, we develop a series of research works for trajectory pattern mining and explore patterns mined for location-based social services. In our study, we present how to collect users' trajectories first. Then, two kinds of mining algorithms are proposed. Finally, we develop a framework for mining location-based social community structures. We briefly introduce each work as follows:

In the first work, we focused on the problem of data collection of trajectory data in a vehicular sensor network where every vehicles are equipped GPS and can communicate with each other in an ad-hoc manner. We proposed a framework MDC to reduce the amount of data transmission and the number of vehicles

reporting their GPS data points. In MDC, model functions are derived to represent the raw GPS data points such that only some coefficients that describe its movements are reported. An in-network aggregation mechanism determines a set of groups and for each group, only one vehicle needs to report traffic data, thereby further reducing the number of simultaneous connections.

In the second work, we proposed a regression-based approach to mine user movement patterns from call detail records in a mobile computing system. Call detail records are viewed as random sample trajectory data, and user movement patterns are represented as movement functions. At first, the call detail records that capture frequent user movement behaviors are extracted. By exploring the spatiotemporal locality of movements, call detail records describing the similar behaviors are clustered. The movement functions can be represented by regression lines to best fit the location and time of call detail records.

In the third work, we proposed an algorithm for discovering trajectory patterns by exploiting trajectory clues. In reality, there are many factors, such as sampling method, sampling frequency and device constraints, will affect the capability of original trajectory data capturing the actual movements. Even if trajectories can only reflect partial movements of a user, they reveal some trajectory clues about the moving behaviors hidden in trajectories. We first propose a clue-similarity to measure how much clue between two trajectories. Based on the clue-similarity, a graph-based clustering algorithm is proposed to group trajectories with similar moving behaviors into the same cluster. At last, for each group, the spatial and temporal information are aggregated into trajectory patterns.

In the fourth work, we targeted at the problem of mining user communities in a location-based social network, where users in the same community have the similar movement behaviors. At the first, trajectory patterns of each user are organized into a probabilistic suffix tree, which is viewed as a trajectory profile of each

user. Inspired by the concept of the edit distance of two sequences, the distance function of two trees is proposed. Finally, in light of the distance of trees, a user communities in a location-based social network are found by clustering users with similar trajectory patterns.



誌 謝

本篇論文雖然列出了六個章節，但是其中最重要的章節隱藏在空白的地方，而你們都是這個最重要章節的主角。回首當初決定念博士班時的那個夏天，只是憑著一股衝動，就毅然地「簽」了下去。但是卻是怎麼也沒料到博士班生涯中將面臨許多令人無法預期的精彩與許多不為人知的苦澀，我想這些是沒有走過這一遭的人無法深刻體會的。

首先，我想要先感謝的是我的指導教授彭文志老師以及李旺謙老師。記得在某一次的聚會上，彭老師對著與會的人員說，他覺得現在的學生最重要的就是 Connect to the world（與世界聯結），而他也使終稟持著這個信念來訓練我。每當有外賓來訪，我總是第一個被推上去與其交談；而每年我也定期的出國，在國際會議上做發表我的論文，在這個過程中，我無意間訓練了與人相處的膽量以及英文溝通能力；這些專業的能力使我在畢業時找工作較一般人來的更加的順利：除了在台灣本地的知名外商公司外，還獲得國外研究機構的面試機會。在飛機上時，我心中萬般激動，只想跟老師說，我終於不負您期待，與這個世界聯結了。另外，在做研究方面，不能不提及的是李旺謙老師的指導。我本身個性大而化之，常沒有注意到研究上的細節以及辨證。在這一點上，李旺謙老師雖然遠在美國，但時常與他有共事的機會。李旺謙老師非常注重問題本身的哲學意義以及解決問題時的辨證細節，所以有多次在跟他報告我的工作時，才剛進入主題便被打回了票。雖然這過程難免感到挫折，但是卻一次又一次的提醒了我做為研究人員應有的思維及素養。因此，在這裡我要誠摯地感謝這兩位對我研究生涯影響最為深刻的導師。另外，在本篇論文中有很多的發表，都是與學弟妹一起共同完成的。在這裡，我想要感謝的與我共同完成論文的學弟妹們。

最後，我想要感謝我的家人爸爸、媽媽、妹妹以及我的女朋友陳鈺玟。很感謝你們總是支持著我，也總是很體諒以及包容我的忙碌。在外面別人總是問我能飛多高，而在家人的口中總是擔心我是否飛得太累。當我每一次挑燈夜戰時，你們不捨的眼神，我都謹記在心，一定不能讓你們失望。也是因為有你們的支持，因此我才能夠不畏懼地面臨一次又一次的挑戰。也是因為有你們認為我是很優秀的，因此我才能夠總是打起精神面對一次又一次的挫折。在畢業典禮上接受全場注目以及撥穗的那一刻，我的心中只想將這份榮耀與你們分享。這一切的一切都要歸功於你們。

謹以本篇論文獻給所有的主角們，沒有你們，這一篇論文是不可能產生的。

Contents

Abstract	iii
Contents	vi
List of Figures	ix
List of Tables	xii
1 Introduction	1
1.1 Model-driven Traffic Data Acquisition in Vehicular Sensor Networks	2
1.2 A Regression-based Approach for Mining User Movement Patterns from Random Sampling Data	3
1.3 CACT: Clustering and Aggregating Clues for Trajectories for Mining Trajectory Patterns	4
1.4 Exploiting Trajectory Profiles for Mining User Communities	5
2 Model-driven Traffic Data Acquisition in Vehicular Sensor Networks	7
2.1 Introduction	7
2.2 System Overview	10
2.2.1 Vehicle Side	10
2.2.2 Server Side	11

2.3	Model Functions and In-Network Aggregation in Vehicle Sides . . .	12
2.3.1	Objective of Model Functions	13
2.3.2	Model Function for Movements of Vehicles	13
2.3.3	Model Function for Speeds of Vehicles	17
2.3.4	In-network Aggregation Mechanism	21
2.4	Statistic Manager in the Server Side	23
2.5	Performance Evaluation	25
2.5.1	Methodology	25
2.5.2	Experiments of MDC and CarTel	26
2.5.3	Sensitivity Analysis	27
2.6	Conclusion	29
3	A Regression-based Approach for Mining User Movement Patterns from Random Sample Data	31
3.1	Introduction	31
3.2	A Regression-based Approach for Mining User Movement Patterns	36
3.2.1	An Overview	36
3.2.2	Algorithm LS: Extracting the Aggregation Movement Sequence	38
3.2.3	Algorithm TC: Clustering Aggregation Movement Records	43
3.2.4	Algorithm MF: Deriving Movement Functions	49
3.2.5	Estimating A User's Location Based on a Movement Function	55
3.3	Performance Evaluation	56
3.3.1	Modeling User Behaviors	56
3.3.2	Experiments of <i>UMP</i> and <i>RUMP</i>	60
3.3.3	Sensitivity Analysis of <i>RUMP</i>	63

3.4	Conclusions	68
-----	-----------------------	----

4 CACT: Clustering and Aggregating Clues of Trajectories for Mining Trajectory Patterns 70

4.1	Introduction	70
4.2	Related Works	77
4.2.1	Similarity Measurements	77
4.2.2	Trajectory Clustering	79
4.2.3	Trajectory Pattern Mining	81
4.3	Problem of Trajectory Pattern Mining	83
4.4	Clue-Aware Trajectory Similarity	85
4.4.1	Characteristics of Trajectories	86
4.4.2	Design of Clue-Aware Trajectory Similarity	87
4.4.3	Properties of Clue-based Similarity Measurements	91
4.5	Clue-Aware Trajectory Clustering Algorithm	92
4.5.1	Design of Clue-Aware Trajectory Clustering Algorithm	93
4.5.2	Running Example for Clustering Discovery in algorithm CATC	100
4.6	Clue-Aware Trajectory Aggregation Algorithm	101
4.6.1	Determine Candidate Line Segments within a Core Set	101
4.6.2	Generate Representative Line Segments within a Cluster	104
4.7	Performance Evaluation	107
4.7.1	Experimental Environment	108
4.7.2	Performance Comparison of Similarity Measurements	110
4.7.3	Performance Comparison of Clustering Algorithms	112
4.7.4	The Impact of Silent Durations for Trajectory Pattern Mining	116

4.7.5	Sensitivity Analysis for CACT	121
4.8	Conclusions	127
5	Exploiting Trajectory Profiles for Mining User Communities	128
5.1	Introduction	128
5.2	The Framework of Mining Communities	130
5.2.1	Constructing Profiles	131
5.2.2	Formulating Distance of Profiles	133
5.2.3	Identifying Community	139
5.2.4	Selecting Representative PSTs	141
5.2.5	Performance Comparison	143
5.2.6	Sensitivity Analysis	144
5.3	Conclusion	145
6	Conclusion	147

List of Figures

1.1	A road map of this chapter	2
2.1	The speed distribution from highway sensors.	9
2.2	The MDC framework.	12
2.3	An illustrative example for algorithm LR.	18
2.4	The kernel regression model by (a) two and (b) three data points.	22
2.5	Cost ratio with (a) grid length and (b) bandwidth varied.	25
2.6	(a) Cost ratio and (b) RMSE with β varied.	28
2.7	(a) Cost ratio and (b) RMSE with ϵ_ℓ varied.	28
2.8	(a) Cost ratio and (b) RMSE with ϵ_s varied.	29
3.1	An example of call detail records	34
3.2	A movement function of a user	35
3.3	An illustrative example of deriving confident movement functions.	54
3.4	A snapshot of a complete movement function $F(t)$	56
3.5	The frequent movement behavior in CarWeb dataset	57
3.6	Performance comparisons of <i>UMP</i> and <i>RUMP</i> on the synthetic dataset	61
3.7	Data utilization of <i>UMP</i> and <i>RUMP</i> on the synthetic dataset	62
3.8	Performance comparisons of <i>UMP</i> and <i>RUMP</i> on the <i>CarWeb</i> dataset	62
3.9	Data utilization of <i>UMP</i> and <i>RUMP</i> on the <i>CarWeb</i> dataset	63

3.10	Execution time for various numbers of movement sequences	64
3.11	Precision ratio of <i>RUMP</i> with varying w	65
3.12	Precision ratio of <i>RUMP</i> with min_freq varied	66
3.13	Precision ratio of <i>RUMP</i> with min_sim varied	67
3.14	Precision ratio with min_var varied	68
3.15	Precision ratio of <i>RUMP</i> with min_sim varied	69
4.1	An example of trajectory pattern mining.	71
4.2	An example of trajectories from CarWeb dataset.	73
4.3	Two sampling methods for collecting trajectories.	76
4.4	Examples for trajectory convoy and moving clusters.	82
4.5	A trajectory pattern.	84
4.6	Overview of our proposed framework CACT.	86
4.7	An illustrative example for clue-aware similarity of T_1 and T_2	90
4.8	An example to show the asymmetric property of CATS.	93
4.9	An illustrative example for CATC.	95
4.10	An example of generating candidate line segments in a core set. . .	104
4.11	An example of tuning representative line segments from other candidate line segments.	105
4.12	Trajectories in <i>CarWeb</i> dataset.	109
4.13	Average error rates with k varied.	112
4.14	Average error rates with varying the distribution of silent periods. .	113
4.15	Combination of CATS and different clustering algorithms.	114
4.16	Comparison of trajectory clustering algorithms.	116
4.17	Comparison of TPM, SPF, and CACT.	118
4.18	Precision and recall of SFP-L, SFP-C, TMP-L, TMP-C and CACT with P_{clue} varied.	120

4.19	Normalized distance of LCSS, EDR, and CATS with P_{clue} , r , and t varied.	122
4.20	Purity and Entropy of CACT with λ varied.	124
4.21	Precision and recall of CACT with ϵ varied.	125
4.22	Precision and recall of CACT with τ varied.	125
4.23	Execution time under (a) short and (b) long trajectories.	126
5.1	Frequent regions by the density-based approach.	131
5.2	An example profile of PSTs.	134
5.3	An execution scenario for the distance of T_1 and T_3 . Only underlined values will be stored in this table.	140
5.4	Comparison of GSP and our approach.	143
5.5	Sensitivity analysis when $MinSup$ varied.	144

List of Tables

3.1	An example of call detail records	33
3.2	Notations used in our algorithms	39
3.3	An example of algorithm LS	42
3.4	An execution scenario of algorithm TC	47
3.5	Data points with their corresponding weights	51
3.6	The parameters used in experiments	60
4.1	Comparison of similarity measurements.	78
4.2	Statistic information about the selected trajectories in CarWeb dataset.	110
4.3	Execution Time (in seconds).	120
5.1	Tree size and error sum of three PSTs	141
5.2	Example of selecting an r-PST.	142



Chapter 1

Introduction

With the advancement of mobile technology, users' locations are easily obtained by the positioning devices embedded in their smart phones. Nowadays, these location information are widely used in various kinds of interesting location-based services, such as Google Latitude and Facebook Places. These services also lead to an increasing amount of users who are willing to contribute their locations information. There will be a huge volume of users' location data which are available to discovery valuable knowledge in near future. Since one's trajectories reflect where he moves and stays in the real world, *trajectory patterns* are one of the most valuable knowledge which can be discovered from his trajectories. These trajectory patterns can be widely used to develop many innovative applications, such as customized business advertisement, personalized services design, traffic flow estimation, urban planning, and so on. This dissertation aims at deriving a series of algorithms for trajectory pattern mining and its application.

Figure 1.1 shows a road map of this dissertation. Specifically, Chapter 2 provides an overview of location data collection, including a discussion of location data properties and an example of a large-scale data collection mechanism in vehicular sensor networks. The location data collected can be then capitalized to mine trajectory patterns of users. According to the different types of location

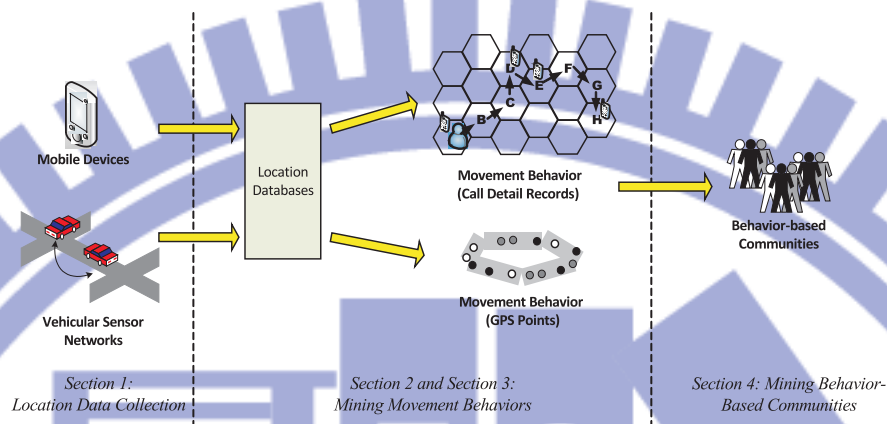


Figure 1.1: A road map of this chapter

data, Chapter 3 and Chapter 4 focuses on mining trajectory patterns by call detail records and GPS points. In light of the concept that two users with the similar movement behaviors may possibly have some social connections on location-based services, implicit user communities can be discovered by grouping users with similar trajectory patterns. Different than the existing social networks with the explicit relations among users, such communities could reveal the implicit connection relations according to their behaviors in the real world, thereby providing more insight of potential interaction among users in location-based social networks. The brief introduction of each section in this chapter is given as follows:

1.1 Model-driven Traffic Data Acquisition in Vehicular Sensor Networks

According to different positioning techniques, location data can be represented into either symbolic or geometric representation. For example, a user's location in a cellular network is represented as a base station identification, which is symbolic representation; a user's location that obtained by GPS is in geometric representa-

tion. We first give an overview to describe the properties of two representations, including granularity, strength and weakness, and so on. Then, we discuss the problem of large-scale data collection in vehicular sensor networks where every vehicles can obtain their locations by GPS and can communicate with each other in an ad-hoc manner. Framework MDC is proposed to reduce the amount of data transmission and the number of vehicles reporting their GPS data points. In MDC, model functions are derived to represent the raw GPS data points such that only some coefficients that describe its movements are reported. An in-network aggregation mechanism determines a set of groups and for each group, only one vehicle needs to report traffic data, thereby further reducing the number of simultaneous connections.

1.2 A Regression-based Approach for Mining User Movement Patterns from Random Sampling Data

Mobile computing systems usually express a user movement trajectory as a sequence of areas that capture the user movement trace. Given a set of user movement trajectories, user movement patterns refer to the sequences of areas through which a user frequently travels. In an attempt to obtain user movement patterns for mobile applications, prior studies explore the problem of mining user movement patterns from the movement logs of mobile users. These movement logs generate a data record whenever a mobile user crosses base station coverage areas. However, this type of movement log does not exist in the system and thus generates extra overheads. By exploiting an existing log, namely, call detail records, this article proposes a Regression-based approach for mining User Movement Patterns (abbreviated as *RUMP*). This approach views call detail records as random sample trajectory data, and thus, user movement patterns are represented as movement

functions in this article. We propose algorithm LS (standing for Large Sequence) to extract the call detail records that capture frequent user movement behaviors. By exploring the spatio-temporal locality of continuous movements (i.e., a mobile user is likely to be in nearby areas if the time interval between consecutive calls is small), we develop algorithm TC (standing for Time Clustering) to cluster call detail records. Then, by utilizing regression analysis, we develop algorithm MF (standing for Movement Function) to derive movement functions. Experimental studies involving both synthetic and real datasets show that *RUMP* is able to derive user movement functions close to the frequent movement behaviors of mobile users.

1.3 CACT: Clustering and Aggregating Clues for Trajectories for Mining Trajectory Patterns

Many research works have been conducted to discover trajectory patterns from a set of trajectory data. Their methods tend to firstly determine hot regions that contain a sufficient number of trajectories, and then mine sequential relationships among these hot regions to discover trajectory patterns. Note that trajectory patterns capture the frequent movement behaviors of a user. Since a user may have multiple movement behaviors, the hot regions determined do not necessarily reflect of the user's movement behaviors. As such, in this chapter, we propose a framework, namely, Clustering and Aggregating Clues of Trajectories (CACT) for identifying movement behaviors of a user. Given a set of trajectories, we intend to cluster trajectories into several groups, where each group represents one movement behavior of a user. In reality, trajectories have the temporal and spatial bias, and silent durations which indicates time durations that no data points are available to describe movements of users. Thus, we employ the clue-aware trajec-

tory similarity to reflect some clues (referred to some data points that are close in both the spatial and temporal domains) between two trajectories. In light of clue similarities among trajectories, in this work we propose a clue-aware clustering algorithm for clustering trajectories that infer similar movement behaviors. Trajectories in the same group are then aggregated to determine hot regions and the corresponding trajectory patterns are derived. Experimental results are conducted on both synthetic and real datasets. Experimental results show that CACT can be more effective in discovering trajectory patterns when compared with existing previous works of mining trajectory patterns.

1.4 Exploiting Trajectory Profiles for Mining User Communities

In the end of this dissertation, we propose a novel community structure where each user community here refer to a group users with similar trajectory patterns. Since two users with similar trajectory patterns implies they are frequently stayed or moved in the same spatial regions, they are likely to interact with each other by some location-based services when they are in nearby regions. To discover such implicit relations, users with similar trajectory patterns should be grouped. To facilitate compare patterns of users, their patters are first organized as tree structures. Inspired by the concept of the edit distance of two sequences, the distance of two trees can be used to represent how similar trajectory patterns of two users are. Finally, user communities can be found by clustering users with similar trees.

The rest of the paper is organized as follows. Trajectory data collection mechanisms are discussed in Chapter 2. Chapter 3 and Chapter 4 proposed algorithms for mining trajectory patterns from CDRs and GPS points, respectively. Chapter

5 described the algorithm for mining implicit user communities. Finally, Chapter 6 concludes with this paper.



Chapter 2

Model-driven Traffic Data Acquisition in Vehicular Sensor Networks

2.1 Introduction

Traffic monitoring is the most important functionality in intelligent transportation systems. Traffic status brings benefits to many applications, such as navigation, route planning, and traffic signal control. To obtain traffic status, one existing method is to deploy static sensors along with roads. Such sensors could be speed detection devices or camera devices [11][36]. However, deploying sensors on all roads is costly, which is not practical for large-scale traffic monitoring.

In recent years, the global position system (GPS) is widely used in technical products, such as navigation devices, GPS loggers, PDAs and mobile phones. Many research and implementation efforts have involved in traffic data collection platforms, which are based on client-server architectures [42][27][50]. In general, such a traffic data collection platform consists of servers and vehicles where each vehicle equips with GPS modules and the wireless communication interfaces, such as 3G or WiFi networks, and the sensed data (e.g., the speed and the position) are sent to the server for traffic monitoring. Prior works focus on

how to estimate traffic status of road networks with GPS data points uploaded and demonstrated the effectiveness of monitoring traffic status by GPS data points uploaded. Clearly, with these GPS data, one could estimate traffic status without any aid of costly sensors for traffic monitoring systems.

To obtain more accurate traffic status from GPS data points, it is expected to have more GPS data points uploaded by vehicles. Assume that each vehicle uploads its GPS data points at a fixed time period. For example, Mobile Millennium Project sets each vehicle to report its position and the corresponding speed every 3 seconds [50]. One challenge issue is that if a significant number of vehicles upload their GPS data points at the same time, the wireless network cannot offer enough network resources for simultaneous network connections. An empirical study of [53] shows that there is a trade-off between data calls and video/voice calls. Hence, when many vehicles establish uplink to upload their GPS data points in bursts, simultaneous network connections decrease the performance of wireless networks, which affects the other services, such as voice and video calls. Note that traffic data usually has the spatial-temporal locality feature, which means that the traffic status on the nearby roads at the nearby time are very similar. Figure 2.1 shows the speed distribution from static sensors deployed on a highway, where the Y-axis is the location of sensors, the X-axis is the time, and the color is the speed reported by the sensor. It can be seen in Figure 2.1, the speed readings of nearby sensors at nearby time are very similar, which shows the spatial-temporal locality. Thus, it is not necessary for each vehicle to report/upload its GPS data points at every time period. Consequently, by exploring the spatio-temporal locality feature, we intend to reduce the amount of GPS data points uploaded and the number of vehicles that need to report their sensing readings to the server.

This paper proposes a framework MDC (standing for Model-based Data Collection) for data collection in traffic monitoring. Since traffic status of roads are

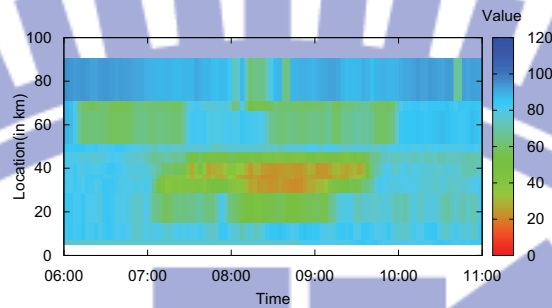


Figure 2.1: The speed distribution from highway sensors.

estimated from GPS data points from time to time, GPS data points are viewed as one kind of sensing data. Most previous studies in sensor data management allow collected data having a certain error with the raw data [10][12]. Same as in prior studies, traffic data collected by MDC are error-bounded to the raw data. The MDC framework is executed at the server and vehicle side collaboratively. In the vehicle side, GPS data points sensed by each vehicle include two attributes: the position and the speed. Given a series of GPS data points, model functions can be derived to represent the raw GPS data points. Hence, each vehicle could report some coefficients that describes its movements instead of reporting all position information. Also, the server will receive coefficients of our proposed model functions for a series of speed readings sensed by vehicles. The challenge issue is how to derive model functions for GPS data points. Since vehicles move along with road segments that are usually a set of line segments, algorithm LR (standing for Liner Regression) is proposed to determine a set of line functions to represent movements of vehicles. By observing the spatial-temporal locality in traffic data, algorithm KR (standing for Kernel Regression) is proposed to derive a set of kernel functions to model a series of speed readings sensed. Moreover, with the spatial-temporal locality feature in traffic data, an in-network aggregation mecha-

nism is proposed to determine a set of groups and for each group, only one vehicle needs to report traffic data, thereby further reducing the number of simultaneous connections. Through the above algorithms and in-network aggregation mechanism, not only the amount of GPS data points uploaded but also the number of vehicles reporting their models are reduced. For the server side, algorithm SM (standing for Statistic Management) is proposed to dynamically adjust sampling rates of vehicles. Extensive experiments are conducted and experimental results shows the effectiveness and the efficiency of our proposed MDC framework.

The rest of this paper is organized as follows. Section 1.2 devises the proposed framework MDC. Section 1.3 and Section 1.4 describes the methodologies in the vehicle side and the server side, respectively. Section 1.5 presents experimental results. Finally, Section 1.6 draws conclusions.

2.2 System Overview

This section presents the framework of MDC. Figure 2.2 shows that MDC consists two components performed at both the vehicle side and the server side. The following sections describe the detailed description of each component.

2.2.1 Vehicle Side

In MDC, each vehicle should sense both the speed and the position readings, and report these readings to the server. The main issue in the vehicle side is to reduce the amount of data transmission to the server. Hence, three modules, *modeling*, *aggregation*, and *communication*, are designed to address this issue. Generally speaking, each vehicle should record its own GPS data points (i.e., the speed and the position) with the corresponding time. Compression skills are used in the modeling module to compress GPS data points. That is, some model functions can be used to represent the whole readings of GPS data points. The

model functions, derived by algorithm LR (standing for Liner Regression) and algorithm KR (standing for Kernel Regression), will be described later. According to the model functions of GPS data points, one could recover the raw readings of GPS data points with a tolerable error bound. With the spatial-temporal locality in traffic data, nearby vehicles may detect the similar traffic status. Hence, these vehicles have similar model functions and in the aggregation module, vehicles communicates with each other and their model functions are aggregated. With the model function derived, the communication module is to report some coefficients or parameters to the server. After receiving the coefficients or parameters about the model functions, the server is able to recover all the reading of GPS data points.

2.2.2 Server Side

The main tasks of the server side are to receive the model functions of vehicles and adjust the number of vehicles reporting their model functions. Given model functions, the server could derive the original readings of vehicles, including the speed and the position of vehicles. According to the recovered readings, the *statistic manager* is built to maintain a statistical structure for traffic data collected. By observing the statistical structure, the server could imply the traffic status of monitoring regions. When the server needs more information due to the error of readings or the change of model functions, the server will issue acquisition to vehicles via the communication module in the vehicle side. The acquisition may indicate the number of vehicles reported and the sampling rates for traffic data collection. The detailed algorithm for traffic data acquisition will presented later. In brief, when the traffic status of monitored regions does not change significantly (e.g., the traffic jams), the server only needs a few vehicles in a traffic jams to report their model functions. On the other hand, if the traffic jam is disappeared, the

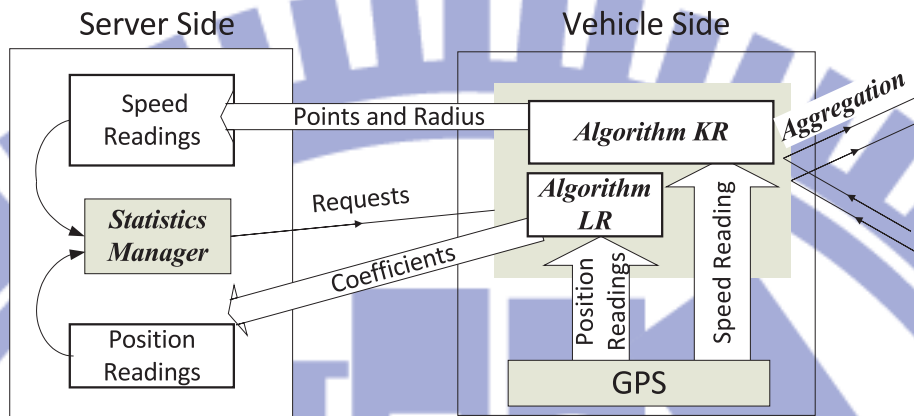


Figure 2.2: The MDC framework.

server thus asks more vehicles to report their model functions to capture the traffic changes. Therefore, from the feedback information of vehicles, the statistic manager can be aware of the spatiotemporal locality hidden in traffic data collected. Similarly, the statistic manager is able to dynamically adjust the sample rates of vehicles to reflect traffic status while still guaranteeing that the readings of traffic data are within the error bound required.

2.3 Model Functions and In-Network Aggregation in Vehicle Sides

This section first presents the objective of model functions in the vehicle side. According to the objective of model functions, algorithm LR, that utilizes linear regression model, is proposed to model position readings of vehicles (i.e., movements of vehicles). To model the speed readings of vehicles, algorithm KR based on kernel regression model is proposed. Finally, an in-network aggregation mechanism is developed.

2.3.1 Objective of Model Functions

The GPS data points of vehicles can be represented as a sequence of points, denoted as $S = \langle (\ell_1, s_1, t_1), (\ell_2, s_2, t_2), \dots, (\ell_n, s_n, t_n) \rangle$, where at time t_i , $\ell_i = (x_i, y_i)$ is the position of vehicles (i.e., the values in the longitude and latitude format), and s_i is the speed readings sensed. With the GPS data points, in the modeling module, we derive some model functions to represent the original raw data such that only model information should be transmitted to the server, thereby reducing the amount of data transmission. The difference between readings derived from the model functions and original readings of vehicles are required to be within a pre-defined error bound. Thus, the root mean square error (abbreviated as RMSE) between the readings derived by models and the original readings are bounded within user-specified error bounds. Given the error thresholds ϵ_ℓ and ϵ_s , the models \hat{f} and \hat{g} , which are derived by our proposed algorithm LR and algorithm KR, should satisfy the following requirements: $RMSE(S, \hat{f}) = \sqrt{\frac{1}{n} \sum_{i=1}^n (\ell_i - \hat{f}(t_i))^2} \leq \epsilon_\ell$ and $RMSE(S, \hat{g}) = \sqrt{\frac{1}{n} \sum_{i=1}^n (s_i - \hat{g}(t_i))^2} \leq \epsilon_s$.

2.3.2 Model Function for Movements of Vehicles

In light of the objective of model functions, this section presents algorithm LR to model movements of vehicles. Since the movements of vehicles are on the road networks that consist of road segments, movements of vehicles can be modeled as a series of line segments that could fit road segments of road networks. Instead of transmitting the detailed position information of vehicles, a series of line segments can be used to approximate movements of vehicles. Thus, the amount of position data reported by vehicles can be reduced.

Background of Linear Regression

Since the positions of a vehicle are relevant to time, we can represent the positions with respect to time into a sequence of position points $(x_1, y_1, t_1), \dots, (x_m, y_m, t_m)$, where x_i and y_i is the longitude and latitude of the vehicle at time t_i . Denote $(X(t), Y(t))$ be the estimated coordinate of a vehicle at time t . We intend to derive two vectors (a_x, a_y) and (b_x, b_y) such that $(X(t), Y(t)) = (a_x, a_y) \times t + (b_x, b_y)$ when t is in the valid time interval $[t_1, t_m]$.

Given a sequence of position points $(x_1, y_1, t_1), \dots, (x_m, y_m, t_m)$, the following matrices are defined:

$$H = \begin{bmatrix} 1 & t_1 \\ 1 & t_2 \\ \vdots & \vdots \\ 1 & t_m \end{bmatrix}, F = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_m & y_m \end{bmatrix}, A = \begin{bmatrix} a_x & a_y \\ b_x & b_y \end{bmatrix}$$

According to linear algebra, we could have $A = (H^T H)^{-1} H^T F$, where the linear regression lines for these position points $(X(t), Y(t)) = (a_x, a_y) \times t + (b_x, b_y)$ are derived and the *RMSE* is minimized.

Since the new position points of vehicles keep generating and are recorded at the vehicle side, one challenge issue is that whether we should use another new line to represent the recent movements or not. Thus, we should justify whether the line derived by recent movements is similar to the original line or not.

Suppose that the coefficient matrices of a linear regression line L are $A = \begin{bmatrix} a_x & a_y \\ b_x & b_y \end{bmatrix}$. The *direction vector* of L , denoted as \vec{v}_L , is represented as (a_x, a_y) . The similarity of two lines could be measured as cosine similarity. Assume that we have direction vectors \vec{v}_{L_1} and \vec{v}_{L_2} . The similarity of two vectors is formulated as $sim(\vec{v}_{L_1}, \vec{v}_{L_2}) = \frac{\vec{v}_{L_1} \cdot \vec{v}_{L_2}}{|\vec{v}_{L_1}| |\vec{v}_{L_2}|}$. The larger value in the cosine similarity between two vectors, the more similar in their lines.

Design of algorithm LR

Given an error threshold ϵ_ℓ , the derived model functions \hat{f} should make sure that the value of $RMSE(S, \hat{f})$ is smaller than ϵ_ℓ . To guarantee that the RMSE is bounded, algorithm LR should detect the variation of movements such that a vehicle can upload the regression line that can best represent the current movement. To achieve this goal, a vehicle should maintain four linear regression lines: the line derived at current time point (denoted as CL), the line uploaded to the centralized server last time (denoted as SL), the backward line (denoted as BL) and the forward line (denoted as NCL). In the beginning, algorithm LR keeps collecting position data points until the number of data points is larger than a threshold N_{min} . It is because that if there are a few position points collected, a new coming point may affect the direction of CL significantly. Thus, a vehicle should collect sufficient amount of data points such that the derived CL can capture the current movement from these data points. Then, the CL will be first sent to the server side and be stored in the both side as SL . Consequently, the server side can use SL to derive the moving behavior of a vehicle. For example, given $N_{min} = 4$, there are four data points in Figure 2.3(a). The CL is first generated by these data points. CL is also stored as SL and CL is uploaded to the server side.

With time passing by, a vehicle collects more and more position points. The movements may become complicated and thus need be described by several linear regression lines rather than one. For example, in Figure 2.3(e), it can be seen that this vehicle makes a turn between d_5 and d_6 , and the original SL in Figure 2.3(a) cannot represent its moving behavior after d_6 . To detect such deviation, a *break point* is used to divide the whole set of data points into two subsets. According to such a break point, we can decide which position points can be used to derive a new CL to represent its current moving behavior precisely. Thus, the server

can capture new behavior by deriving SL according to the new regression line uploaded.

Algorithm LR incrementally maintains a potential break point b to verify whether the moving direction changes or not. Specifically, suppose that the current position points involving in CL are (d_1, d_2, \dots, d_k) where d_k is a new incoming position point. According to a potential break point b , we can derive a backward regression line BL for $(d_1, \dots, b = d_j)$ and a forward regression line NCL for (d_j, \dots, d_k) with their direction vectors \vec{v}_{BL} and \vec{v}_{NCL} . Given a threshold β , if $\text{sim}(\vec{v}_{BL}, \vec{v}_{NCL}) > \beta$, the moving behavior described by position points before and after b does not change tremendously so that b is set to be the new coming position point d_k . Otherwise, the position of b does not change since position points before b and that after b may represent different moving behavior. Given the error bound for trajectories ϵ_T , if the $\text{RMSE}((d_1, \dots, d_k), SL) > \epsilon_T$, these position points represent different moving behavior than SL does. In this case, a vehicle should upload the forward regression line NCL to the server to set the CL and SL to be NCL to guarantee the model satisfying the error bound. The detailed algorithm is shown below:

Consider an illustrative example in Figure 2.3. The threshold β is shown in terms of angle. Once the angle between two vectors are larger than the given angle, the similarity between these two vectors are larger than β . In the beginning, in Figure 2.3(a), the CL is uploaded to the server side and also stored as SL . From d_4 to d_5 , the potential break point b keeps going forward to be d_4 and d_5 since $\text{sim}(\vec{V}_{BL}, \vec{V}_{NCL}) > \beta$. With the new data points d_6 coming, as shown in Figure 2.3(b), the break point b keep being d_5 since $\text{sim}(\vec{V}_{BL}, \vec{V}_{NCL}) < \beta$. However, the line NCL is unnecessary to be uploaded to the server side. Figure 2.3(c) shows that CL is still like SL . Until d_8 is generated, Figure 2.3(e) shows that RMSE between d_6, d_7, d_8 and SL is large enough. Thus, as shown in Figure 2.3(d), the

Algorithm 1: Algorithm LR

INPUT : Error Bound: ϵ_ℓ ; Splitting Threshold: β ; Minimum Number of Data Points: N_{min}
OUTPUT: The uploaded line: SL

```
1 while there is a new incoming data point  $d_k$  do
2   if the number of points  $\leq N_{min}$  then
3      $SL \leftarrow (0, 0)$ ;
4     Collect data points;
5      $b \leftarrow d_k$ ;
6   else
7      $CL \leftarrow$  regression line by  $\{d_1, \dots, d_k\}$ ;
8      $BL \leftarrow$  regression line by  $\{d_1, \dots, b = d_j\}$ ;
9      $NCL \leftarrow$  regression line by  $\{b = d_j, \dots, d_k\}$ ;
10    if  $RMSE(\{d_1, \dots, d_k\}, SL) > \epsilon_\ell$  then
11       $SL \leftarrow CL \leftarrow NCL$ ;
12       $\{d_1, \dots, d_k\} \leftarrow \{d_j, \dots, d_k\}$ ;
13       $b \leftarrow d_k$ ;
14      Upload  $SL$ 
15    if  $sim(\vec{v}_{BL}, \vec{v}_{NCL}) > \beta$  then
16       $b \leftarrow d_k$ ;
```

forward regression line NCL is assigned to be the new current regression line CL and uploaded to the server side as new SL .

2.3.3 Model Function for Speeds of Vehicles

When a query indicates the data acquisition at one query range specified, by checking whether line segments of vehicles are within the query range or not, we could quickly identify those vehicles that are within the query range. Then, for those vehicles, we could ask for their models of speed readings. The amount of speed readings reported could be reduced by our proposed model functions and our in-network aggregation mechanism, which will be presented later. The distributions of speed readings may vary. For example, the speed readings collected

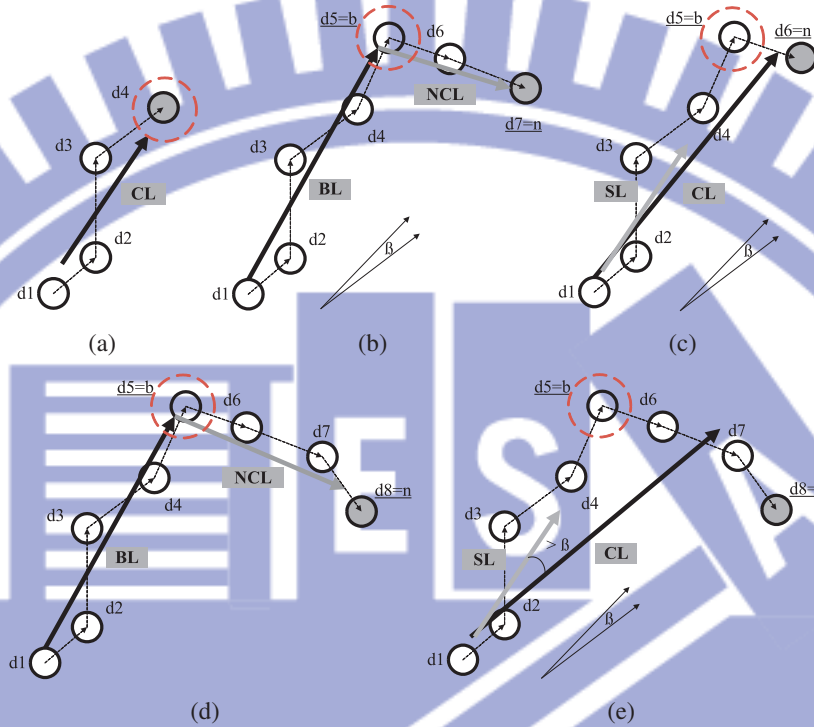


Figure 2.3: An illustrative example for algorithm LR.

in urban areas are likely to be different from those readings in downtown areas. Thus, without a priori knowledge on the distribution of speed readings, the kernel regression function is able to select some data points from a set of given data points to derive one curve that fits all data points given. In the following sections, the kernel regression function is briefly presented and then we propose algorithm KR to derive kernel regression functions for speed readings of vehicles.

Background of Kernel Regression

Given a set of data points, the idea of kernel regression is to select a set of data points, and use these selected data points to estimate other data points. For example, suppose that there are n data points $\{(s_1, t_1), \dots, (s_n, t_n)\}$. From n data points, we select k data points as a set of feature data points, denoted as $\Gamma = \{(\sigma_1, \tau_1), \dots, (\sigma_k, \tau_k)\}$, where σ_i denotes the speed reading at time τ_i . By utilizing the feature data points, we could derive any speed readings in the set of

original data points. Consider that given a time t , the speed reading at time t is derived as follows:

$$\hat{s}_t = \frac{\sum_{j=1}^k \sigma_j \times K_\alpha(t, \tau_j)}{\sum_{j=1}^k K_\alpha(t, \tau_j)} \quad (2.1)$$

, where $K_\alpha(t, \tau_j)$ is the kernel with its kernel radius α_j .

In kernel regression, the kernel $K_\alpha(t, \tau_i)$ is viewed as the weight value for estimating the speed at time t . Clearly, if the time of the select data point is close to the time t , the weight of this selected data point should be larger. The determination of weights for each feature point is the main theme of the kernel regression. Generally speaking, for feature data point (σ_i, τ_j) , the weight of this feature data point is proportional the distance between τ_j and t . If the distance between τ_j and t is larger than one threshold, called kernel radius α , the weight is set to zero. Otherwise, the weight is set to the value of $K_\alpha(t, \tau_j)$, where $K_\alpha(t, \tau_j)$ is the kernel function. When t is exactly the same to τ_j , the value of $K_\alpha(t, \tau_j)$ should be larger. Otherwise, the value of $K_\alpha(t, \tau_j)$ will be decreased as the distance between t and τ_j increases. There are many kernel functions available. Generally, Gaussian kernel function can be selected where the Gaussian kernel is formulated as $K_\alpha(t, \tau_j) = e^{-\frac{(t-\tau_j)^2}{2\alpha^2}}$. According to research works in [21], the accuracy of kernel regression is highly dependent on the selection of kernel radius. As a result, our experimental results will demonstrate the impact of kernel radius.

Design of Algorithm KR

As pointed out early, traffic data usually has spatial-temporal locality. Hence, speed readings are usually similar if their time are close. With this locality feature, utilizing kernel regression to model a series of readings is able to significantly reduce the amount of data. Clearly, a series of readings is transformed as a kernel regression function. The challenge issue is how to determine the set of selected data points for each kernel function while guaranteeing that RMSE is smaller than

Algorithm 2: Algorithm KR

INPUT : Error bound: ϵ_s ; Minimal Number of Data Points: N_{min}
OUTPUT: The radius: α ; Feature Points: Γ

```
1 while there is a new incoming data point  $d_k$  do
2   if  $RMSE(C, \Gamma \cup \{d_k\}) < \epsilon_I$  or  $|\Gamma| < N_{min}$  then
3      $\Gamma \leftarrow \Gamma \cup \{d_k\}$ ;
4   else
5      $rate \leftarrow rate_{min}$ ;
6      $rmse \leftarrow \epsilon_s$ ;
7     while  $rmse > \epsilon_s$  AND  $n \geq |\Gamma|$  do
8        $n \leftarrow rate \times |\Gamma|$ ;
9        $\Gamma \leftarrow$  Sample  $n$  data points;
10       $\alpha \leftarrow$  kernel radius by ROT according to  $\Gamma$ ;
11       $C \leftarrow$  kernel regression model with  $\alpha, S$ ;
12       $rmse \leftarrow RMSE(\Gamma, C)$ ;
13       $rate \leftarrow rate + rate_{inc}$ ;
14      if  $rmse \geq \epsilon_s$  then
15        Upload( $\alpha, \Gamma$ );
16      else
17         $\Gamma \leftarrow d_k$ ;
```

ϵ_s .

To build kernel regression functions, we should collect a sufficient number of data points. One predefined parameter N_{min} is used. When the number of readings are larger than N_{min} , the procedure of deriving kernel regression functions is performed. Specifically, with a number of data points, we could randomly sample some data points as a set of feature data points. The number of data points sampled is controlled by sampling rate $rate_{min}$. With a larger sampling rate, more data points are selected as feature data points. With the feature data points selected, rule-of-thumb (ROT) method [51] is performed to determine the initial kernel radius. Based on the ROT, the bandwidth $\hat{\alpha}$ for the kernel function $K_{\hat{\alpha}}(\cdot)$ could be computed using the following equation: $\hat{\alpha}_i = \left(\frac{4}{3}\right)^{-1/5} |S|^{-1/5} SD$, where

SD is the standard deviation of the feature points. According to these selected feature points and the derived kernel radius, we can exploit kernel regression to generate a curve fitted them. Once satisfying the error bound, a vehicle can upload the feature points Γ and the corresponding kernel radius to the server such that the same curve can be derived by them in the server side. Otherwise, the sampling rate is increased to select more data points for deriving kernel regression functions. When a new data point is arrived, the new data point will be verified whether the current kernel regression function is valid or not. If the current kernel regression function cannot estimate the reading of the new data point, algorithm KR will derive another new kernel regression function by accumulating enough number of new data points.

Consider the data points $\{d_1, d_2, \dots, d_5\}$ in Figure 2.4(a). Suppose that the error bound ϵ_s is 2.5, $rate_{min}$ is 40% and $rate_{inc}$ is 20%. Initially, we sample $5 \times 40\% = 2$ points which are d_1 and d_4 . Figure 2.4(a) shows the results by two kernels centered at d_1 and d_4 . Since the RMSE of this curve is 2.71 which is larger than ϵ_s , then the sampling rate is increasing to $40\% + 20\% = 60\%$ such that we sample $5 \times 60\% = 3$ points, d_1, d_2 and d_4 , to apply kernel regression. It can be verified that the RMSE of the derived curve is 2.41 smaller than ϵ_s and these three data points are selected as feature data points and the corresponding radius is recorded.

2.3.4 In-network Aggregation Mechanism

Since vehicles nearby usually have similar readings in terms of their movements and the speed readings, in this section, we further develop an in-network mechanism to reduce the number of vehicles reported. With the above model functions, each vehicle has its own model functions both in the movements and speed readings. In our proposed mechanism, there are two phases: *nearby grouping and*

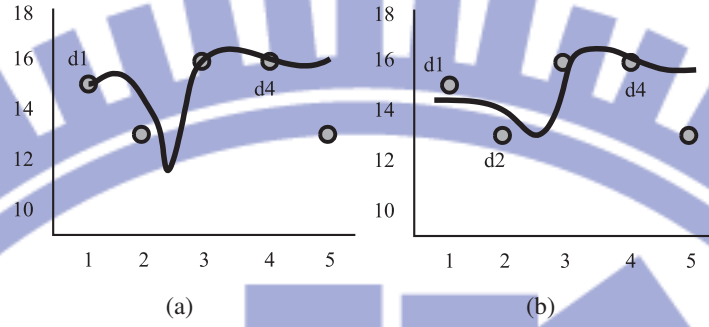


Figure 2.4: The kernel regression model by (a) two and (b) three data points.

aggregator selection and *aggregation*. The details of each phase are described as follows:

Phase 1: Nearby Grouping and Aggregator Selection

Since nearby vehicles are likely to have similar model functions in their speed readings, these vehicles have higher probability to be in the same group. Among these nearby vehicles, one should select an aggregator to perform aggregation of model functions and report some model functions to the server. Due to the deployment of wireless access points, an aggregator should be one vehicle with higher connection probability of accessing wireless network. To consider the occurring time of successful connections, each vehicle will maintain a variable $P_{connect} = I_k \times \frac{1}{2} + I_{k-1} \times \frac{1}{2^2} + \dots$ where I_{k-i} is 1 (or 0) if the i -th connection before the current connection is (or not) successful. $P_{connect}$ provides a metric to evaluate how often the recent successful connection happens. It can be seen that $P_{connect}$ gives higher weight to the current successful connection and the weight of the successful connection will exponential decay according to their occurring time. That is, $P_{connect}$ will be larger if successful connections occur frequently, and vice versa. For two vehicles in the same group, the vehicle with larger $P_{connect}$ will be the aggregator.

Phase 2: Aggregation

In this phase, the aggregator will further obtain model functions of vehicles in the same group to determine whether their model functions could be aggregated or not. Thus, we need one evaluation for the aggregation of model functions. Let two models be M_A and M_B with valid time interval $[t_s^A, t_e^A]$ and $[t_s^B, t_e^B]$. Let $T = [t_s^A, t_e^A] \cup [t_s^B, t_e^B]$, the sampling RMSE between M_A and M_B is $SR(M_A, M_B) = \sqrt{\frac{1}{|T|} \sum_{t_i \in T} |M_A(t_i) - M_B(t_i)|}$

Based on the sampling RMSE, M_A and M_B can be aggregated if $SR(M_A, M_B)$ is smaller than the error bound. Among all models which sampling RMSE are mutually smaller than the error bound, the model with the fewest number of data points will be selected and reported to the server. By this approach, the number of vehicles which need to be reported their models can be reduced, thereby further reducing the amount of data transmissions.

2.4 Statistic Manager in the Server Side

This section describes algorithm SM (standing for Statistic Manager) to dynamically adjust the sample rates of vehicles. After receiving model functions of vehicles, the server is aware of the position and the speed readings of vehicles. The server could further adjust the sampling rate according to the traffic status collected. For example, the traffic jam may last for a while and those vehicles in the traffic jam could reduce the sampling rate since the traffic status does not significantly change in the near future. Thus, the statistic manager in the server side maintains a histogram for speed readings of vehicles. Since the traffic data has the spatial-temporal locality, the whole region can be divided into several grids and for each grid, the histogram from speed readings of vehicles in that grid is constructed. The histogram construction refers to construct the best histogram re-

stricted to a space bound that reflects the data distribution most accurately under a given error measure, which can be achieved by [22].

Based on the histogram of grids, the sampling rates of vehicles can be dynamically adjusted. Assume that the sampling rate ζ_i is grid G_i . Intuitively, ζ_i is set larger if the data distribution of a grid G_i does not change significantly. Two approaches are proposed to detect the variation of histograms. The first approach is to monitor the deviation of current and history histograms of this grid. There are several proposed approaches to compare the similarity of two histograms. The χ^2 -test (Chi-square test) is the most commonly used one since it is not affected by the amount of total history data. Thus, once the current histogram can pass the χ^2 -test from the history one, which means that the current data distribution and the history one are the same, the sampling rate ζ_i can thus be reduced. Otherwise, the sampling rate will be increased to guarantee the accuracy of collected data. The second approach is to monitor the result of in-network aggregation. The aggregator will report not only the model functions but also the number of vehicles, which models are aggregated by the aggregator, in its nearby range. Suppose that aggregator A_j has $N(A_j)$ vehicles in its group. The value of $N(A_j)$ reflects the spatial-temporal locality among its group. The larger number of vehicles in A_j 's nearby region, the higher the spatiotemporal locality. Therefore, the statistical manager can verify the ratio of the number aggregated models and the total number of vehicles to increase or decrease the value of ζ_i . Specifically, the ζ_i will be increased when $\frac{\sum_{A_j \in G_i} N(A_j)}{TN_i} < \zeta_i$ where TN_i represents the total number of vehicles in grid G_i , and vice versa.

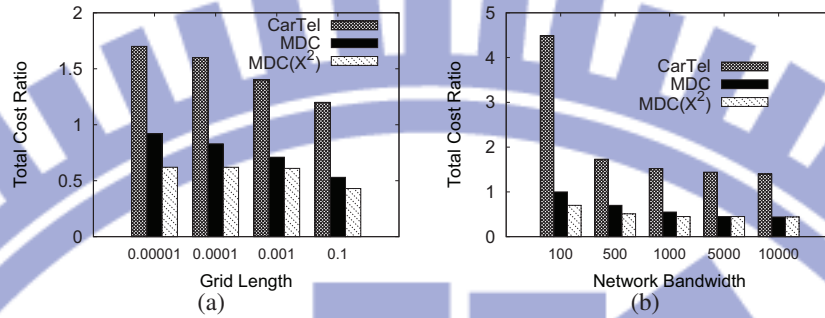


Figure 2.5: Cost ratio with (a) grid length and (b) bandwidth varied.

2.5 Performance Evaluation

This section presents extensive experimental results to show the performance of the proposed framework MDC.

2.5.1 Methodology

In the following experiments, the well-known traffic simulator [4] is used for environment simulation. To reflect reality, the map of San Francisco in the real world are used. To simulate the intermittent connectivity, 20 access points are set into the map. In the server side, the map will be divided into several grids according to the parameter *cell length*. The server will build a histogram for each grid by the data upload from vehicles within it. In the simulation, there are 250 vehicles with three classes of speed of a vehicle, say slow, medium and fast. Each data point generated by a vehicle is composed of four attributes, say longitude, latitude, collected time stamp and the speed.

In the following experiments, RMSE and cost ratio are the performance measurements. Specifically, *RMSE* is used to measure the error between the derived models and the raw data points. To guarantee the quality of collected data by MDC, the value of RMSE is required not to exceed the given error bounded. The

cost ratio represents the proportion between the amount of transmitted of models and the effective data which are derived from models by the server. A lower cost ratio represents that MDC can reduce the amount of data uploaded more effectively.

For the comparison purpose, we implement a centralized data collection platform CarTel [27]. In CarTel, sensor data will be assigned different priority weights to determine the corresponding transmission orders. To determine the transmission order of data, a vehicle should send a summary report about the sensed data to the server. Then, by this report, the server decides the transmission order and sends it to this vehicle. In our experiments, the priority is decided by the road identifications where the sensed readings are generated. That is, some roads will be viewed more important than others. The experimental time is set as 2000 time units. The default parameters are: $\beta = 0.8$, $\epsilon_\ell = 0.25$, $\epsilon_s = 6$, wireless bandwidth as 5000 bytes and cell length as 0.01.

2.5.2 Experiments of MDC and CarTel

In the experiments, we compare the cost ratios of MDC and CarTel with different grid length and network bandwidth settings. In our experiments, we test the server with and without χ^2 -test, which denote as $\text{MDC}(\chi^2)$ and MDC, respectively.

Figure 2.5(a) shows that the cost ratio of CarTel is higher than MDC in all cases, which indicates that MDC can effectively reduce the amount of data transmitted. It is because the smaller grid length leads to more histograms needed to be constructed. Therefore, CarTel spends more cost to decide the transmission cost to obtain the raw data and build histograms for grids than MDC. On the other hand, comparing with MDC, $\text{MDC}(\chi^2)$ can further reduce the cost ratio. It is because $\text{MDC}(\chi^2)$ can adaptively adjust the sampling frequency such that transmission cost is then reduced.

Figure 2.5(b) shows the cost ratio with the network bandwidth varied. The network bandwidth affects the amount of data transmitted simultaneously. That is, the larger bandwidth an AP owns, the more simultaneous connections an AP can afford. In CarTel, a vehicle transmit its raw data to the server. Thus, it can be seen that the cost ratio of CarTel is large when the bandwidth is small since data re-transmission will occurs frequently. On the other hand, MDC can reduce more cost ratio than CarTel does. Note that the cost ratio of our approach is almost 1 when the bandwidth is 100. It can show that even data re-transmission happens, the amount of data transmitted by MDC can be controlled within a reasonable range.

2.5.3 Sensitivity Analysis

In MDC, two error bounds, say ϵ_ℓ and ϵ_s , should be specified to guarantee the quality of data collection. Moreover, in algorithm LR, an addition parameter β is used to distinguish whether the current regression line and the upload one follow the same moving behavior or not. In this section, we discuss the impact of these thresholds to MDC with and without aggregation mechanism.

Impact of β

Figure 2.6 shows the impact of β in terms of cost ratio and RMSE. With β increasing, the cost ratio increases and the RMSE decreases. There is a tradeoff between cost ratio and RMSE. When the β is close to 1, the number of derived regression lines increases. Thus, the cost ratio increases since many regression lines should be uploaded. Since many regression lines are used, the moving behavior of a vehicle can be precisely described such that the RMSE decreases. In this experiment, choosing $\beta = 0.6$ can take a good balance between cost ratio and RMSE. On the other hand, in Figure 2.6(b), when $\beta = 0.6$, in-network aggregation mechanism

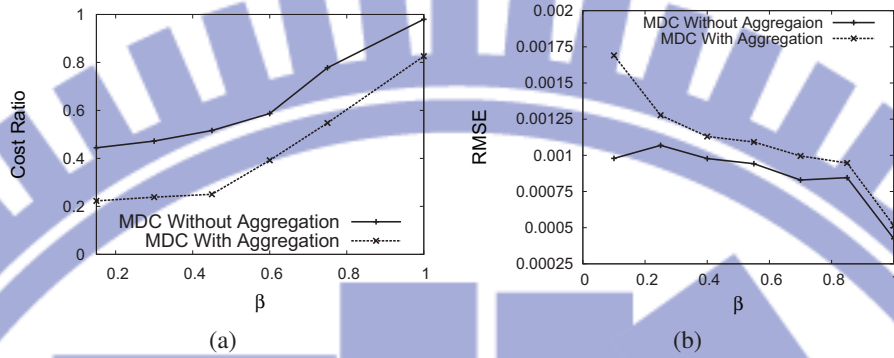


Figure 2.6: (a) Cost ratio and (b) RMSE with β varied.

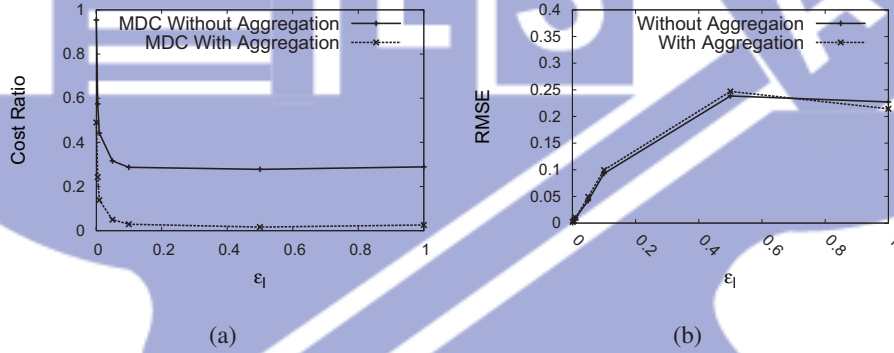


Figure 2.7: (a) Cost ratio and (b) RMSE with ϵ_ℓ varied.

can reduce the cost ratio at most 20% but the RMSE only increase slightly. It shows that our aggregation approach can not efficiently reduce the cost ratio but also preserve the accuracy of uploaded models.

Impact of ϵ_ℓ and ϵ_s

Figure 2.7 shows the results with ϵ_ℓ varied. Figure 2.7(a) shows two cost ratios of MDC. The cost ratio converges to a fixed value in both MDC with and without aggregation. It can be seen that the cost ratio of MDC with aggregation can achieve almost 0.05, which shows that the linear regression model we selected is properly

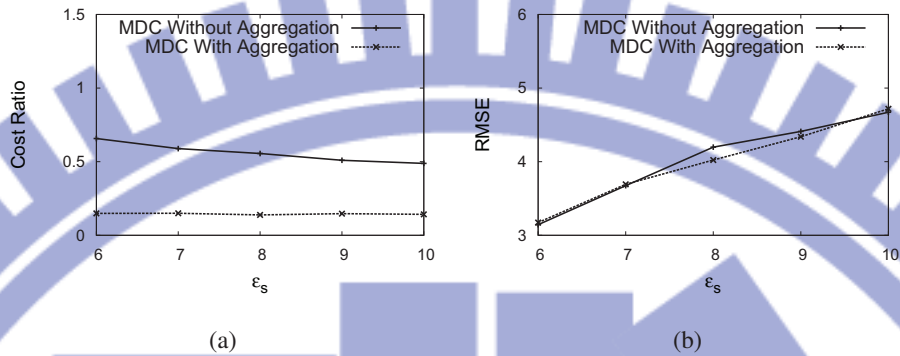


Figure 2.8: (a) Cost ratio and (b) RMSE with ϵ_s varied.

used to represent the trajectories of vehicles. Figure 2.7(b) shows that the RMSE will first increase when $0 < \epsilon_\ell < 0.4$ and keep constant when $\epsilon_\ell > 0.4$. Moreover, RMSE of both MDC with and without aggregation can be bounded by ϵ_ℓ , which shows the quality of collected data can be guaranteed. On the other hand, Figure 2.8 shows the results with ϵ_s varied. Similar to the results above, the cost ratio of MDC with aggregation can achieve a better performance while the RMSE is almost the same as that of MDC without aggregation.

Overall, we can conclude that MDC can not only reduce the data amount efficiently but also guarantee the quality of data collection.

2.6 Conclusion

In traffic monitoring applications, a naive approach of data collection is that each vehicle uploads the position and speed readings to the centralized server immediately when the data points are generated. However, this approach is not scalable enough for a large-scale system since it needs a considerable amount of network bandwidth and simultaneous connection. This paper proposed MDC to reduce the amount of readings during data collection. MDC performs at both vehicle and

server sides collaboratively. In the vehicle side, Algorithm LR and Algorithm KR can use linear and kernel regression model to reduce the size of transmitted data, while the errors between the raw data the derived model are bounded. Moreover, in-network aggregation is employed to further reduce the amount of data while guaranteeing the quality of data. In the server side, both pull and push approaches are exploited to collect data from vehicles. Experimental studies show that MDC can collect traffic data efficiently and effectively.

Chapter 3

A Regression-based Approach for Mining User Movement Patterns from Random Sample Data

3.1 Introduction

Mobile services, such as navigation services, mobile search and location-aware services, are becoming very popular. These wireless communication systems enable users to access various kinds of information from anywhere at any time. A mobile computing system usually expresses a user movement trajectory as a sequence of areas in which the mobile user moves¹. In this article, we aim at mining user movement patterns for a mobile user. Thus, given a user's set of movement trajectories, user movement patterns refer to the sequences of areas that this user frequently travels. User movement patterns can be used to improve system performance, such as designing personal paging area [59], and developing data allocation strategies [55][46][47], querying strategies [32], and navigation services [34][19].

To discover user movement patterns in a mobile computing system, the meth-

¹This article defines a unit of a area as the coverage area of one base station. For ease of presentation, we simply use a base station identification to represent the corresponding coverage area.

ods proposed in previous studies require movement logs to record the movements of mobile users. For example, in [46][47], when a mobile user moves from the coverage area of base station i to the coverage area of base station j , a hand-off procedure is performed to smoothly switch communication channels between base stations. Meanwhile, the movement log generates a movement pair (i, j) . However, the movement log is not an existing log of mobile systems and needs some overheads to generate during handoff procedures. Hence, generating movement logs for all mobile users leads to increased storage costs and decreases the performance of mobile computing systems. Therefore, prior works are not practical for mobile computing systems due to the overhead of generating movement logs. In fact, mobile computing systems generate call detail records (abbreviated as CDR) when a mobile user makes or receives phone calls [41]. Table 5.2.4 shows an example of call detail records, where Uid represents the identification of a user making or receiving calls, and Cellid represents the corresponding base station that serves that user. Time information (i.e., date and time) is recorded in the CDR². Table 5.2.4 shows that the CDRs of a mobile user contain both spatial (i.e., base station identification) and temporal information (i.e., date and time). Since CDRs reflect the movement behaviors of users, this article addresses the problem of mining user movement patterns from an existing log of CDRs, thereby reducing the overhead of generating a movement log.

Figure 1 shows some trajectories of one user, where the dashed line represents one real trajectory of this user and the regions with mobile phones indicate that the user is receiving or making phone calls. This user's calling behavior is captured in the log of CDRs, and Table 5.2.4 shows the CDR log. Figure 3.1 shows that CDRs are data points that are randomly sampled from trajectories and the corresponding

²The real call detail records analyzed in this study were provided by Taiwan mobile service providers, and we only extracted some useful attributes of call detail records to mine user movement patterns.

Uid	Date	Time	Cellid
1	Day 1	07:30	A
1	Day 1	09:32	D
1	Day 1	09:49	E
1	Day 1	13:50	H
1	Day 2	08:50	C
1	Day 2	09:50	E
1	Day 2	14:00	H
1	Day 3	07:15	A
1	Day 3	09:02	C
1	Day 3	09:30	D
1	Day 4	12:30	W
1	Day 4	12:52	X
1	Day 4	13:30	Y

Table 3.1: An example of call detail records

locations of these CDRs are scattered over the mobile computing environment. As a result, mining user movement behaviors from CDRs is a challenging task. Given these random sample data points, we aim to derive movement functions that are close to real user trajectories. We refer these movement functions as movement patterns due to that movement functions reflect the frequent movement behavior of users. Figure 3.2 shows the movement function of a user for the example above.

This article proposes a novel approach, called *RUMP* (standing for Regression-based approach for User Movement Patterns), to mine user movement patterns from CDRs. Given a set of data points, the main objective of regression analysis is to derive a regression function that minimizes the sum of distances between the function derived and data points. In this approach, call detail records are viewed as data points, while the regression functions derived are regarded as movement functions. However, not all call detail records should be involved in mining user movement patterns. Without carefully selecting CDRs, user movement patterns cannot reflect the frequent movement behaviors of mobile users. On the other

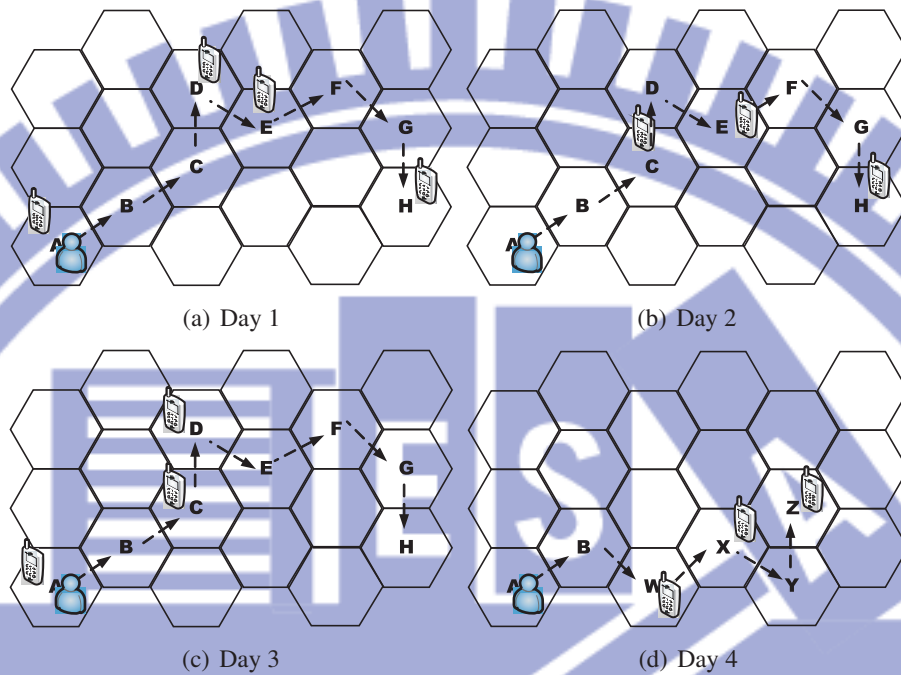


Figure 3.1: An example of call detail records

hand, CDRs should be fully utilized for mining user movement patterns since only limited information is available in the CDR logs. Thus, several issues remain to be addressed to efficiently utilize CDRs for mining user movement patterns.

- **Extracting frequent movement behaviors from CDRs**

As mentioned before, user movement patterns refer to the frequent movement behaviors of mobile users. However, the CDR logs not only contain frequent user movement behaviors, but also include infrequent movement behaviors. For example, a user usually goes to his office and is back to his home every weekday (as Figure 3.1(a), (b) and (c) shows), and occasionally takes a trip (as Figure 3.1(d) shows). The frequent movement behavior is the trajectory from his home to his office, whereas a trip is an infrequent movement behavior. Since regression analysis is sensitive to these infrequent CDRs, they should be eliminated. In other words, the call detail records that capture the frequent movement behaviors of

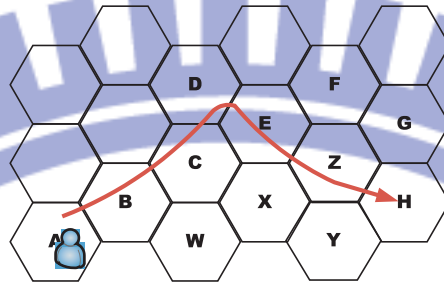


Figure 3.2: A movement function of a user

users should be extracted. To extract the frequent movement behaviors of mobile users, we develop algorithm LS (standing for Large Sequence) to extract base stations whose coverage areas are frequently visited by users.

- **Determining the number of regression functions**

Once CDRs that capture the frequent movement behaviors have been extracted, it is necessary to determine how many regression functions are needed. If only one regression function is derived, it may not be very close to the frequent user movement behavior. Thus, given a set of call detail records of the frequent movement behavior, clustering techniques can be used to divide call detail records into several groups. The number of groups is viewed as the number of regression functions. The movement trajectories of mobile users generally follow spatio-temporal locality (i.e., if the time interval between two consecutive calls of a mobile user is small, the mobile user is likely to have moved nearby). Therefore, the feature of spatio-temporal locality in algorithm TC (standing for Time Clustering) can be used to group the call detail records with a close occurrence time.

- **Deriving movement functions**

Location identification techniques typically use one of two location models: the geometric model and the symbolic models [37]. The geometric model specifies the location in n-dimensional coordinates (typically n=2 or 3). The symbolic model, however, uses logical entities to describe location. This article represents

the location of mobile users in CDRs using the symbolic model (i.e., the base station identification). To derive movement functions of a mobile user, the location of call detail records in the symbolic model must be transformed into the geometric model. Then, with the cluster results obtained, we develop algorithm MF (standing for Movement Function) for each cluster. This algorithm utilizes weighted regression analysis to derive the corresponding movement functions of a user.

The *RUMP* approach consists of a series of algorithms that tackle the various issues described above. This study evaluates *RUMP* performance using both synthetic and real datasets. Sensitivity analysis is conducted on several design parameters. Experimental results show that *RUMP* is able to efficiently and effectively derive user movement patterns that capture the frequent movement behaviors of mobile users.

The rest of this article is organized as follows. Section 2.2 then devises algorithms for mining user movement patterns. Section 2.3 presents performance results. Finally, Section 2.4 draws conclusions.

3.2 A Regression-based Approach for Mining User Movement Patterns

This section develops a regression-based approach (i.e., *RUMP*) consisting of a sequence of algorithms to mine user movement patterns. First, Section 3.1 provides an overview of *RUMP*, and the following sections present details of Algorithm LS, TC, and MF.

3.2.1 An Overview

Given a log of CDRs, the goal of this article is to derive movement functions that closely reflect the frequent movement behaviors of mobile users. Due to that

CDRs are random samples, the timestamps of CDRs are not likely to be the same even if a user follows the same movement behavior. Consequently, a basic time slot is defined as a time interval. For example, if call detail records whose occurrence time is within the time interval of one time slot, these CDRs are associated with the same time slot. Therefore, these CDRs are further put in a movement record defined as follows:

Definition: Movement Record: A movement record is defined as a set of pairs $(BS_i : N_i)$, where BS_i is a base station and N_i is the number of occurrence counts of BS_i in call detail records whose occurring times are within the same time slot.

Assume that one time unit has its time interval of 6:00am to 10:00am. From Table 5.2.4, in Day 1, we could have one movement record that includes $\{A:1, D:1, E:1\}$ since the occurrence time of three call detail records (i.e., A, D, and E) is within the time interval (i.e., 6:00am to 10:00am). With the definition of movement records, a movement sequence is defined as follows:

Definition: Movement Sequence: A movement sequence MS_i , denoted by $\langle MR_{i,1}, MR_{i,2}, MR_{i,3}, \dots, MR_{i,\varepsilon} \rangle$, is an ordered sequence of ε movement records, where $MR_{i,j}$ is the movement record at time slot j in MS_i and ε is an adjustable parameter.

The length of a time slot determines the granularity of user movement patterns in terms of time. Same as in [29], the value of ε indicates that a movement pattern may re-appear. Thus, the value of ε depends on the periodicity of a user. Table 3.2 are notations used in our article. The overall procedure for mining movement patterns is outlined as follows:

Execution Steps in RUMP

Step 1. (Extracting the Aggregation Movement Sequence) In this step, call detail records are converted into w movement sequences, where w is an adjustable window size for the recent movement sequences being considered. *Algorithm LS* discovers an aggregation movement sequence, in which each movement record contains frequent areas that a user appears.

Step 2. (Clustering Movement Records) According to the aggregation movement sequence derived, we further develop *algorithm TC* to cluster movement records whose time slots are close.

Step 3. (Deriving Movement Functions) We then use regression techniques to derive the corresponding movement functions for each group in Step 2.

CDRs only reflect the fragmented movement behaviors of mobile users. Thus, the *RUMP* approach uses regression techniques to derive movement functions which are close to the frequent movement behaviors of mobile users. Due to the nature of regression techniques, without the proper determination of call detail records, user movement functions derived cannot capture the frequent movement behaviors of mobile users. On the other hand, call detail records should be fully utilized to mine user movement patterns since only limited information is available in CDRs. In the following subsections, each algorithm is presented in detail.

3.2.2 Algorithm LS: Extracting the Aggregation Movement Sequence

In this article, a user movement trajectory is represented as a sequence of base station identifications (hereafter, we use "base station" for short). Hence, call detail records are converted into movement sequences. With a set of movement sequences, algorithm LS determines an Aggregation Movement Sequence (abbreviated as *AMS*) and uses it to represent the frequent movement behaviors of a

Definition	Notation
Number of movement sequences	w
Movement sequence i	MS_i
Movement record at time slot j in MS_i	$MR_{i,j}$
A large movement sequence	LMS
Large movement record at time slot i	LMR_i
An aggregation movement sequence	AMS
Aggregated movement record at time slot i	AMR_i
A time projection sequence of AMS	TP_{AMS}
A clustered time projection sequence of AMS	CTP_{AMS}

Table 3.2: Notations used in our algorithms

user. Intuitively, AMS is a sequence of movement records that have frequent base station and their corresponding counts at each time slot. At each time slot, a frequent base station in this article refers to a base station which a user appears more than min_freq times among movement sequences. The min_freq is given to quantify frequent base stations. As pointed out early, counts for frequent base stations should also be determined. Thus, before deriving AMS, a large movement sequence (abbreviated as LMS) is a sequence of frequent base stations and we use LMS to compute the similarity between LMS and each movement sequence. In light of similarity measurements obtained, we are able to identify those movement sequences capturing the frequent moving behavior of users and aggregate them as AMS.

Definition: Large Movement Record: Given a set of movement sequences MS_1, MS_2, \dots, MS_w and a threshold min_freq , a large movement record at time slot t is denoted as LMR_t and LMR_t contains a set of base stations whose occurrence count in the set of movement records at time slot t (i.e., $MR_{1,t}, MR_{2,t}, \dots, MR_{w,t}$) is larger or equal to min_freq .

Given five movement sequences in Table 3.3, if min_freq is set to 2, LMR_4 is $\{D, F\}$ since both D and F have their occurrence count equal to min_freq . Large movement records demonstrate the frequent movement behavior of a user at a specific time slot. After obtaining large movement records at each time slot, a large movement sequence LMS is thus a sequence of large movement records, which is denoted as $LMS = \langle LMR_1, LMR_2, \dots, LMR_\varepsilon \rangle$. Consequently, LMS indicates the frequent moving behavior of users.

Once a large movement sequence LMS is determined, we should further formulate the similarity between movement sequences and LMS to identify whether a movement sequence is the frequent movement behavior of a user or not. The conventional similarity measurements, such as Cosine similarity [52] and extended Jaccard coefficient, cannot be applied for the similarity measurement because they can only deal with scalar vectors with no missing values. Movement sequences and LMS are sequences of sets of base stations, not a sequence scalar values. Moreover, empty sets are allowed in movement sequences and LMS . As such, we formulate the similarity between a movement sequence (e.g., MS_i) and LMS as the closeness between movement records $MR_{i,j}$ and LMR_j , denoted by $C(MR_{i,j}, LMR_j)$. $C(MR_{i,j}, LMR_j)$ compares the set of base stations in $MR_{i,j}$ with the frequent base stations in LMR_j . $C(MR_{i,j}, LMR_j)$ is formulated as $\frac{|\{x \in MR_{i,j} \cap LMR_j\}|}{|\{y \in MR_{i,j} \cup LMR_j\}|}$, and returns the normalized value in $[0, 1]$. The larger the value of $C(MR_{i,j}, LMR_j)$, the more closely $MR_{i,j}$ resembles LMR_j . For example, assume that $LMR_j = \{a, b, c, d\}$, $MR_{x,j} = \{b, e\}$ and $MR_{y,j} = \{a, b, c, d, e\}$. It can be verified that the value of $C(MR_{x,j}, LMR_j)$ is $\frac{1}{5}$ and the value of $C(MR_{y,j}, LMR_j)$ is $\frac{4}{5}$. Therefore, $MR_{y,j}$ is more similar to LMR_j . Based on the similarity between movement records and large movement records, the similarity measure of movement sequences MS_i and LMS is formulated as $sim(MS_i, LMS) = \sum_{j=1}^{\varepsilon} |MR_{i,j}| * C(MR_{i,j}, LMR_j)$. Given a threshold value

min_sim , for each movement sequence MS_i , if $sim(MS_i, LMS) \geq min_sim$, the movement sequence MS_i is identified as a similar movement sequence. Consider the example in Table 3.3. It can be verified that $sim(MS_1, LMS) = 1 * \frac{1}{2} + 1 * \frac{1}{1} + 0 + 1 * \frac{1}{2} + 1 * \frac{0}{1} = 2$. Further, $sim(MS_2, LMS) = 3$, $sim(MS_3, LMS) = 2$, $sim(MS_4, LMS) = 3$, and $sim(MS_5, LMS) = \frac{1}{2}$. Assuming that min_sim is 2, MS_1 , MS_2 , MS_3 and MS_4 are recognized as similar movement sequences.

After identifying similar movement sequences, these similar movement sequences are aggregated as one AMS in which frequent base stations and their associated counts are determined. An aggregation movement sequence is defined as follows:

Definition: Aggregation Movement Sequence: The aggregate movement sequence is denoted as $AMS = \langle AMR_1, AMR_2, \dots, AMR_\epsilon \rangle$, where AMR_j is an aggregated movement record that contains frequent base stations, which are the same in large movement record LMR_j and their occurring counts accumulated from movement records at time slot j of similar movement sequences.

Consider the AMR_1 of AMS in Table 3.3 as an example. from those similar movement sequences, the occurrence count of A in AMR_1 is calculated as the sum of the count of A in $MR_{1,1}$, that in $MR_{3,1}$ and that in $MR_{4,1}$ (i.e., $14+1+1 = 16$). Following the same procedure, we could have $AMS = \langle \{A : 16, B : 1\}, \{A : 3\}, \phi, \{D : 2, F : 3\}, \{H : 2\} \rangle$ shown in Table 3.3.

Time Complexity Analysis: Given w movement sequences with ϵ time slots, the complexity of algorithm LS can be expressed as $O(\epsilon w)$. The complexity involved in calculating large movement records is $O(\epsilon w)$, while that of extracting frequent movement sequences is $\epsilon * w * O(1) = O(\epsilon w)$. As a result, the overall time complexity of algorithm LS is $O(\epsilon w)$. Thus, algorithm LS is of polynomial time complexity.

	1	2	3	4	5
MS_1	A:14	A:2		F:1	I:2
MS_2			C:8	C:1, D:1, F:1	H:1, G:4
MS_3	A:1	C:1		D:1	H:1
MS_4	A:1, B:1	A:1	F:9		
MS_5	B:4	D:4	H:1		A:1, B:2
LM_S	{A, B}	{A}		{D, F}	{H, I}
AMS	{A:16, B:1}	{A:3}	ϕ	{D:2, F:3}	{H:2}

Table 3.3: An example of algorithm LS

Algorithm 3: Algorithm LS

Input: w movement sequences with length ε , two threshold: min_freq and min_sim

Output: aggregation movement sequence AMS

```

1: begin
2: for  $j \leftarrow 1$  to  $\varepsilon$  do
3:   for  $i \leftarrow 1$  to  $w$  do
4:     begin
5:        $LMR_j \leftarrow$  frequent 1-itemset of  $MR_{i,j}$ ; //by  $min\_freq$ 
6:     end
7:   for  $i \leftarrow 1$  to  $w$  do
8:     begin
9:        $match \leftarrow 0$ ;
10:    for  $j \leftarrow 1$  to  $\varepsilon$  do
11:      begin
12:         $C(MR_{i,j}, LMR_j) \leftarrow |x \in MR_{i,j} \cap LMR_j| / |y \in MR_{i,j} \cup LMR_j|$ ;
13:         $match \leftarrow match + |MR_{i,j}| * C(MR_{i,j}, LMR_j)$ ;
14:      end
15:    if  $match \geq min\_sim$  then
16:      accumulate the occurring counts of items in the aggregation
      movement sequence;
17:    end

```

3.2.3 Algorithm TC: Clustering Aggregation Movement Records

As pointed out early, the movement trajectories of mobile users generally follow spatio-temporal locality (i.e., if the time interval between two consecutive calls of a mobile user is small, the mobile user is likely to have moved nearby). Accordingly, aggregation movement records in AMS could be clustered into several groups if their corresponding time slots are close. To facilitate the presentation of this paper, only time information (i.e., time slots) is extracted from AMS. Thus, time projection sequence of AMS is defined as follows:

Definition: Time Projection Sequence: A *time projection sequence* of AMS is expressed as $TP_{AMS} = \langle \alpha_1, \dots, \alpha_n \rangle$, where $AMR_{\alpha_j} \neq \{\}$ and $\alpha_1 < \dots < \alpha_n$.

A time projection sequence is a sequence of time slots in which the corresponding movement records are not empty. Algorithm TC then uses the time projection sequence to cluster close time slots. The cluster result of algorithm TC is represented as a clustered time projection sequence defined as follows:

Definition: Clustered Time Projection Sequence: A *clustered time projection sequence* of TP_{AMS} , denoted by CTP_{AMS} , is represented as $\langle CL_1, CL_2, \dots, CL_k \rangle$, where the i -th group CL_i is the time slots of the clustered movement records, and k is an integer such that $1 \leq k \leq \varepsilon$.

Given AMS obtained in Step 1, TP_{AMS} is then easily determined. By exploring the feature of spatial-temporal locality, algorithm TC generates a clustered time projection sequence of AMS (i.e., CTP_{AMS}). Each cluster in CTP_{AMS} contains close time slots. Those movement records with their time slots being clustered preserve the feature of spatio-temporal locality. Therefore, the objective of clustering is to bound the variance of time slots in each group with a given threshold (i.e., min_var).

The variance of a group CL_i is defined as $Var(CL_i) = \frac{1}{m} \sum_{k=1}^m (n_{i,k} - \frac{1}{m} \sum_{j=1}^m n_{i,j})^2$

, where $n_{i,j}$ represents the j -th time slots of movement records in CL_i and the total number of movement records in CL_i is m . Algorithm TC generates a clustered time projection sequence CTP_{AMS} such that $Var(CL_i) \leq min_var$ for all clusters CL_i .

To achieve the objective of clustering, algorithm TC first starts coarsely clustering TP_{AMS} into several marked clusters using a value δ . The initial value of δ is set to ϵ and δ then decreases by one for each round. Thus, in the beginning, there is only one cluster. Dividing clusters with a variance larger than min_var increases the number of clusters. In algorithm TC, unmarked clusters refer to clusters that do not need to be refined, whereas marked clusters refer to clusters that should be further partitioned. For each cluster CL_i , if $Var(CL_i)$ is smaller than min_var , the cluster CL_i is unmarked. Otherwise, δ decreases by 1 and algorithm TC re-clusters the time slots in unmarked clusters with the updated value of δ . Algorithm TC partitions TP_{AMS} iteratively until no marked cluster remain or until $\delta = 1$. If there are no marked clusters, CTP_{AMS} is generated. Otherwise, if there are still marked clusters with their variance values larger than min_var , algorithm TC continues to finely partition these marked clusters so that the variance for every marked cluster is constrained by the threshold value of min_var .

When the value of δ is 1, the time slots of movement records in a marked cluster generally follow a sequence of consecutive integers such that the variance of marked clusters is still larger than min_var . This situation results in loss of spatio-temporal locality. For example, given movement records with a sequence of consecutive time slots 1, 2, 3, 4, 5, 6, and 7, though the differences of consecutive time slots are small, the location of a user at time slot 1 and that at time slot 7 are probably far from each other. To deal with this problem, this cluster must be further partitioned into smaller clusters. The variance of each refined cluster should be smaller than min_var . Moreover, to guarantee that no time slots of each

refined clusters are as close as possible, the total variance of the refined clusters should be minimized. To derive the optimal method for further partitioning, the following lemma is derived:

Lemma: *Given a cluster that has a sequence of consecutive integers $1, 2, 3, \dots, n$ and a positive integer k , the optimal method to minimize the sum of variance in each cluster and divide this cluster into k clusters is to partition it into k sub-clusters each with a size of $\lceil \frac{n}{k} \rceil$.*

Proof:

Suppose that $\langle 1, 2, 3, \dots, n \rangle$ is divided into k sub-clusters: $\langle 1, \dots, t_1 \rangle$, $\langle t_1 + 1, \dots, t_2 \rangle$, ..., $\langle t_{k-1} + 1, \dots, n \rangle$. Let $t_0 = 0$, $t_k = n$, and $Var_i = Var(\langle t_{i-1} + 1, \dots, t_i \rangle)$. Our goal is to find the cutting points (i.e., t_1, t_2, \dots , and t_{k-1}) to minimize $f = \sum_{i=1}^k Var_i$.

The variance remains the same constant for a sequence of consecutive integers with the same length. For example, consider two clusters with two sequences of consecutive time slots: $\langle 1, 2, 3, 4, 5 \rangle$ and $\langle 7, 8, 9, 10, 11 \rangle$. It can be verified that $Var(\langle 1, 2, 3, 4, 5 \rangle) = Var(\langle 7, 8, 9, 10, 11 \rangle)$. Since $Var(\langle 1, 2, \dots, n \rangle) = \frac{1}{12}(n^2 - 1)$, we have $f = \sum_{i=1}^k Var_i = \frac{1}{12} \sum_{i=1}^k ((t_i - t_{i-1})^2 - 1)$.

To minimize $f = \sum_{i=1}^k Var_i$, the cutting points t_1, t_2, \dots, t_{k-1} are derived by letting the first derivatives be zero.

$$\begin{cases} \frac{\partial f}{\partial t_1} = 4t_1 - 2t_2 - 2t_0 = 0 \\ \frac{\partial f}{\partial t_2} = 4t_2 - 2t_3 - 2t_1 = 0 \\ \dots \\ \frac{\partial f}{\partial t_{k-1}} = 4t_{k-1} - 2t_k - 2t_{k-2} = 0 \end{cases}$$

Thus, we can have the following terms:

$$\begin{cases} t_1 = \frac{t_0 + t_2}{2} \\ t_2 = \frac{t_1 + t_3}{2} \\ \dots \\ t_{k-1} = \frac{t_{k-2} + t_k}{2} \end{cases}$$

Using substitution, we have

$$\left\{ \begin{array}{l} t_1 = \frac{1}{2}t_2 \\ t_2 = \frac{2}{3}t_3 \\ \dots \\ t_{k-1} = \frac{k-1}{k}t_k \end{array} \right.$$

Therefore, we can get:

$$\left\{ \begin{array}{l} t_1 = \frac{1}{k}n \\ t_2 = \frac{2}{k}n \\ \dots \\ t_{k-1} = \frac{k-1}{k}n \end{array} \right.$$

From the derivation above, the optimal way to divide $\langle 1, 2, 3, \dots, n \rangle$ into k sub-clusters is to divide $\langle 1, 2, 3, \dots, n \rangle$ into k sub-clusters each with size of $\lceil \frac{n}{k} \rceil$.

Q.E.D

This lemma provides a guideline for partitioning a marked cluster that has a sequence of consecutive time slots into smaller clusters. Since the value of k is not known in advance, the value of k is initially set 2, and then increases in each iteration. In each iteration, a marked cluster is evenly divided into k sub-clusters, each with size of $\lceil \frac{n}{k} \rceil$, and the variance of each sub-cluster is tested. If the variance of a sub-cluster is smaller than min_var , the procedure terminates. Otherwise, the value of k is increased by 1 and the marked cluster will be further refined into smaller sub-clusters.

Consider the execution scenario in Table 3.4, where the time projection sequence is $TP_{AMS} = \langle 1, 2, 3, 4, 5, 9, 10, 14, 17, 18, 20 \rangle$. The initial cluster is $\langle 1, 2, 3, 4, 5, 9, 10, 14, 17, 18, 20 \rangle$. Given $min_var = 1.6$, algorithm TC first roughly partitions TP_{AMS} into three clusters. Table 3.4 shows that two marked clusters (i.e., $\langle 1, 2, 3, 4, 5 \rangle$ with $Var(\langle 1, 2, 3, 4, 5 \rangle) = 2$ and $\langle 14, 17, 18, 20 \rangle$ with $Var(\langle 14, 17, 18, 20 \rangle) = 4.69$ are determined because the variance values of these two clusters are larger than 1.6. Then, δ is reduced to 2, and these two marked clusters are re-examined. In the following run, the previous cluster $\langle 14, 17, 18, 20 \rangle$ is divided into two clusters, i.e., $\langle 14 \rangle$ and $\langle 17, 18, 20 \rangle$ in this run. Since $Var(\langle 14 \rangle) = 0 < 1.6$ and $Var(\langle 17, 18, 20 \rangle) = 1.56 < 1.6$,

Run	δ	min_var	Clusters
0	20	1.6	$\langle 1, 2, 3, 4, 5, 9, 10, 14, 17, 18, 20^* \rangle$
...
1	3	1.6	$\langle 1, 2, 3, 4, 5 \rangle^*$, $\langle 9, 10 \rangle$, $\langle 14, 17, 18, 20 \rangle^*$
2	2	1.6	$\langle 1, 2, 3, 4, 5 \rangle^*$, $\langle 9, 10 \rangle$, $\langle 14 \rangle$, $\langle 17, 18, 20 \rangle$
3	1	1.6	$\langle 1, 2, 3, 4, 5 \rangle^*$, $\langle 9, 10 \rangle$, $\langle 14 \rangle$, $\langle 17, 18, 20 \rangle$
4	0	1.6	$\langle 1, 2, 3, 4, 5 \rangle^*$, $\langle 9, 10 \rangle$, $\langle 14 \rangle$, $\langle 17, 18, 20 \rangle$
5	0	1.6	$\langle 1, 2, 3 \rangle$, $\langle 4, 5 \rangle$, $\langle 9, 10 \rangle$, $\langle 14 \rangle$, $\langle 17, 18, 20 \rangle$

Table 3.4: An execution scenario of algorithm TC

these two clusters remain unmarked. Following the same procedure, algorithm TC partitions marked clusters until δ equals 1. Run 4 in Table 3.4 shows that $\langle 1, 2, 3, 4, 5 \rangle$ is still a marked cluster with $Var(\langle 1, 2, 3, 4, 5 \rangle) = 2$. Therefore, algorithm TC finely partitions $\langle 1, 2, 3, 4, 5 \rangle$. The value of k is initially set at 1. Since $Var(\langle 1, 2, 3, 4, 5 \rangle) = 2.5$ is larger than min_var (i.e., 1.6), k increases to 2. Then, $\langle 1, 2, 3, 4, 5 \rangle$ is divided into $\langle 1, 2, 3 \rangle$ and $\langle 4, 5 \rangle$. Of these two clusters (i.e., $\langle 1, 2, 3 \rangle$ and $\langle 4, 5 \rangle$), the $\langle 1, 2, 3 \rangle$ cluster has the larger variance and thus $\langle 1, 2, 3 \rangle$ is compared with the value of min_var . Since the $Var(\langle 1, 2, 3 \rangle) = 0.67 < 1.6$, algorithm TC stops the clustering process. Finally, a CTP_{AMS} is generated as $\langle 1, 2, 3 \rangle$, $\langle 4, 5 \rangle$, $\langle 9, 10 \rangle$, $\langle 14 \rangle$, $\langle 17, 18, 20 \rangle$.

Time Complexity Analysis: Algorithm TC is of polynomial time complexity. Let TP_{AMS} have n numbers. Algorithm TC needs $O(n)$ to divide TP_{AMS} into clusters from line 5 to 15. From line 17 to line 25, assume that there are still s clusters with m numbers to be refined. Since k is at most m , we have $s * O(m)$ to run the clustering process. The worst case occurs when estimating the time complexity of algorithm TC. In the worst case (i.e., $m = n$), the overall time complexity of algorithm TC is at most $O(n)$.

Algorithm 4: Algorithm TC

Input: Time projection sequence: TP_{AMS} , thresholds: min_var

Output: Clustered time projection sequence: CTP_{AMS}

```
1: begin
2:  $\delta \leftarrow \varepsilon$ ;
3:  $CL_1 \leftarrow TP_{AMS}$ ;
4: Mark  $CL_1$ ;
5: while there exist marked clusters and  $\delta \geq 2$  do
6:   begin
7:   for each marked clusters  $CL_i$  do
8:     if  $Var(CL_i) \leq min\_var$  then
9:       begin
10:        unmark  $CL_i$ ;
11:       end
12:      $\delta \leftarrow \delta - 1$ ;
13:   for all marked clusters  $CL_i$  do
14:     group the numbers whose differences are within  $\delta$  in  $CL_i$ ;
15:   end
16:
17: if there are marked clusters then
18:   begin
19:   for each marked cluster  $CL_i$  do
20:      $k = 2$ ;
21:     repeat
22:        $k \leftarrow k + 1$ ;
23:     until the variance of each group  $\leq min\_var$ 
24:   end
25: end
26: end
```

3.2.4 Algorithm MF: Deriving Movement Functions

Given the aggregation movement sequence AMS devised by algorithm LS and its clustered time projection sequence CTP_{AMS} generated by algorithm TC, algorithm MF is able to derive a sequence of movement functions able to estimate the frequent movement behaviors of mobile users. For each cluster, we need to derive *confidence movement functions*. Then, *linkage movement functions* are determined to link confidence movement functions among clusters. Finally, a movement function $F(t)$ is derived and represented as $\langle U_0(t), E_1(t), U_1(t), E_2(t), \dots, E_k(t), U_k(t) \rangle$, where $E_i(t)$ is the confidence movement function in cluster CL_i of CTP_{AMS} and $U_i(t)$ is the linkage movement function from $E_i(t)$ to $E_{i+1}(t)$.

Deriving Confidence Movement Functions

For each cluster CL_i of CTP_{AMS} , the confidence movement function of a mobile user, expressed as $E_i(t) = (\hat{x}_i(t), \hat{y}_i(t), TI_i)$, is derived. In this case, $\hat{x}_i(t)$ (respectively, $\hat{y}_i(t)$) is a movement function in x-coordinate axis (respectively, in y-coordinate axis) and the confidence movement function is valid for the time interval indicated in TI_i .

Without loss of generality, let CL_i be $\langle t_1, t_2, \dots, t_n \rangle$, where t_j denotes one of the time slots in CL_i for $j = 1, 2, \dots, n$. AMR^i contains frequent base stations with their corresponding counts in the i -th time slot of AMS . To derive movement functions, the location of base stations should be converted from the symbolic model into the geometric model through a map table that indicates the coordinates of base stations and is provided by telecompanies. Hence, given AMS and CTP_{AMS} , for each cluster of CTP_{AMS} , the geometric coordinates of frequent base stations can be derived along with their corresponding counts and represented as $(t_1, x_1, y_1, w_1), (t_2, x_2, y_2, w_2), \dots, (t_n, x_n, y_n, w_n)$ where t_i is the corresponding time slot, x_i (respectively, y_i) is the x-coordinate (respectively,

y-coordinate) of the base station, and w_i is the number of phone calls a mobile user has made at this base station. Accordingly, for each cluster of CTP_{AMS} , a weighted regression analysis is able to derive the corresponding confidence movement function.

Given a set of data points, the goal of regression analysis is to derive the best estimated curve with the minimal sum of least square errors [26]. One aggregation movement sequence is generated in Step 1, which calculates the appearance counts of base stations. Thus, based on the appearance counts of base stations, we can derive curves closer to those base stations with larger appearance counts. This is because the more calls a user makes at a base station, the more confidence we have that this mobile user frequently appears in the coverage area of this base station. Another advantage of utilizing weighted regression analysis is that in a real scenario of mobile computing systems, the base station that serves to a user is not always the nearest base station. This is because other base stations nearby will cover the nearest base station when it becomes overloaded. However, the scenario above does not always happen. The appearing counts of other base stations will be fewer than that of the nearest base station. Therefore, weighted regression analysis makes it possible to derive curves close to base stations with higher appearance counts.

Given data points within a cluster, this article considers the derivation of the $\hat{x}(t)$. An m -degree polynomial function $\hat{x}(t) = a_0 + a_1t + \dots + a_mt^m$ is derived to approximate the movement behavior along x -coordinate axis. Given the data points $(t_1, x_1, y_1, w_1), (t_2, x_2, y_2, w_2), \dots, (t_n, x_n, y_n, w_n)$, the regression coefficients $\{a_0, a_1, \dots, a_m\}$ are then selected to minimize the residual sum of squares $\epsilon_x = \sum_{i=1}^n w_i e_i^2$, where $e_i = (x_i - (a_0 + a_1t_i + a_2(t_i)^2 \dots + a_m(t_i)^m))$. The value of m is application dependent, and must be smaller than the number of data points. The value of m is proportional to the precision of the fitting curve. Since $\hat{x}(t)$

t_i	ID	x_i	y_i	w_i
1	A	1	1	16
1	B	1	2	1
2	A	1	1	1
4	D	4	2	2
4	F	3	3	3
5	H	5	3	2
7	K	6	3	4
9	F	3	3	10
10	E	4	3	1

Table 3.5: Data points with their corresponding weights

is obtained by matrix operations, the matrix size is thus the dominant factor in regression performance. However, the impact of weighted regression analysis on execution time is not significant in this article since the maximal value of m is usually small. When the value of m is small, the execution time of regression analysis is acceptable. Therefore, according to the number of data points available, the value of m should be set as large as possible.

For ease of presentation, the following terms are defined:

$$\mathbf{H} = \begin{bmatrix} 1 & t_1 & (t_1)^2 & \dots & (t_1)^m \\ \dots & \dots & \dots & \dots & \dots \\ 1 & t_n & (t_n)^2 & \dots & (t_n)^m \end{bmatrix}, \mathbf{a}^* = \begin{bmatrix} a_0 \\ \dots \\ a_m \end{bmatrix}, \tilde{\mathbf{b}}_x = \begin{bmatrix} x_1 \\ \dots \\ x_n \end{bmatrix}, \mathbf{e} = \begin{bmatrix} e_1 \\ \dots \\ e_n \end{bmatrix}^T,$$

$$\mathbf{W} = \begin{bmatrix} w_1 & & & & \\ & \dots & & & \\ & & & & w_n \end{bmatrix}.$$

By solving the equation $(\sqrt{\mathbf{W}\mathbf{H}})^T(\sqrt{\mathbf{W}\mathbf{H}})\mathbf{a}^* = (\sqrt{\mathbf{W}\mathbf{H}})^T\sqrt{\mathbf{W}}\tilde{\mathbf{b}}_x$, \mathbf{a}^* can be derived such that the value of ϵ_x is minimized³. This leads to $\hat{x}(t) = a_0 + a_1t + \dots + a_mt^m$. $\hat{y}(t)$ can be derived following the same procedure. As a result, for each cluster of CTP_{AMS} , the confidence movement function $E_i(t) = (\hat{x}(t), \hat{y}(t), [t_1, t_n])$ of a mobile user can be devised.

For example, let $AMS = \langle \{A : 16, B : 1\}, \{A : 1\}, \phi, \{D : 2, F :$

³For the proof, see Appendix A.

3}, \{H : 2\} > and the coordinates of A, B, D, F and H be (1, 1), (1, 2), (4, 2), (3, 3) and (5,3), respectively. Given AMS and $CTP_{AMS} = \langle 1, 2, 4, 5 \rangle$, it is possible to obtain data points with their weights, as Table 3.5 shows. By setting m to 3, the 3-degree polynomial $\hat{x}(t) = a_0 + a_1t + a_2t^2 + a_3t^3$ is derived. The coefficients a_0, a_1, a_2 and a_3 are determined by a regression curve that minimize the residual sum error. That is, $\mathbf{a}^* = (a_0 \ a_1 \ a_2 \ a_3)^T$ must be determined. Since there are six data points with their corresponding time

slots of 1, 1, 2, 4, 4 and 5, $\mathbf{H} = \begin{bmatrix} 1 & 1 & (1)^2 & (1)^3 \\ 1 & 1 & (1)^2 & (1)^3 \\ 1 & 2 & (2)^2 & (2)^3 \\ 1 & 4 & (4)^2 & (4)^3 \\ 1 & 4 & (4)^2 & (4)^3 \\ 1 & 5 & (5)^2 & (5)^3 \end{bmatrix}$ is then calculated. The

weights of the data points are 16, 1, 1, 2, 3 and 2, respectively. Hence, $\sqrt{\mathbf{W}}$ is a diagonal matrix with the diagonal entries of $[\sqrt{16}, \sqrt{1}, \sqrt{1}, \sqrt{2}, \sqrt{3}, \sqrt{2}]$. Table 3.5 shows that $\tilde{\mathbf{b}}_x = (1 \ 1 \ 1 \ 4 \ 3 \ 5)^T$. By solving the equation $(\sqrt{\mathbf{W}}\mathbf{H})^T(\sqrt{\mathbf{W}}\mathbf{H})\mathbf{a}^* = (\sqrt{\mathbf{W}}\mathbf{H})^T\sqrt{\mathbf{W}}\tilde{\mathbf{b}}_x$, we can get $\mathbf{a}^* = (2.333 \ -2.133 \ 0.867 \ -0.066)^T$. Therefore, $\hat{x}(t) = 2.333 - 2.133t + 0.867t^2 - 0.066t^3$ is devised to predict the x coordinate-axis of the mobile user from $t = 1$ to $t = 5$. Similarly, $\tilde{\mathbf{b}}_y = (1 \ 2 \ 1 \ 2 \ 3 \ 3)^T$ is then determined from Table 3.5. By solving the normal equation $(\sqrt{\mathbf{W}}\mathbf{H})^T(\sqrt{\mathbf{W}}\mathbf{H})\mathbf{a}^* = (\sqrt{\mathbf{W}}\mathbf{H})^T\sqrt{\mathbf{W}}\tilde{\mathbf{b}}_y$, we can get $\mathbf{a}^* = (2.529 \ -2.386 \ 1.021 \ -0.105)^T$. We can obtain $\hat{y}(t) = 2.529 - 2.386t + 1.021t^2 - 0.105t^3$. Figure 3.3 shows that the confidence movement functions, where the circle point indicates the location of a base station with its corresponding weight and the solid line is the curve derived by algorithm MF. The confidence movement function closely resembles actual movement behavior, demonstrating the advantage of utilizing regression analysis to mine user movement patterns.

Algorithm 5: Algorithm MF

Input: AMS and clustered time projection sequence CTP_{AMS}

Output: A list of movement functions

$F(t) = \langle E_1(t), U_1(t), E_2(t), \dots, E_k(t), U_k(t) \rangle$

```
1: begin
2:  $F(t) = \langle \rangle$ ;
3: for  $i = 1$  to  $k - 1$  do
4:   begin
5:   doing regression on  $CL_i$  to generate  $E_i(t)$ ;
6:   doing regression on  $CL_{i+1}$  to generate  $E_{i+1}(t)$ ;
7:    $t_1$  =the last time slot in  $CL_i$ ;
8:    $t_2$  =the first time slot in  $CL_{i+1}$ ;
9:   using inner interpolation to generate  $U_i(t) = (\hat{x}_i(t), \hat{y}_i(t), (t_1, t_2))$ ;
10:  insert  $E_i(t), U_i(t)$  and  $E_{i+1}(t)$  in  $F(t)$ ;
11:  end
12: if  $1 \notin CL_1$  then
13:   generate  $U_0(t)$  and Insert  $U_0(t)$  into the head of  $F(t)$ ;
14: if  $\varepsilon \notin CL_k$  then
15:   generate  $U_k(t)$  and Insert  $U_k(t)$  into the tail of  $F(t)$ ;
16: end
```

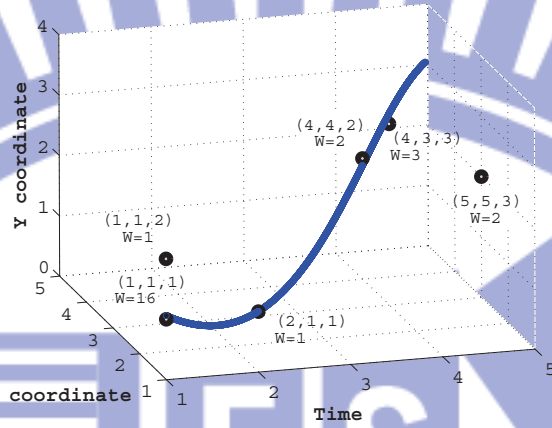


Figure 3.3: An illustrative example of deriving confident movement functions.

Deriving Linkage movement Functions

Given AMS and a cluster of $CTP_{AMS} = \langle CL_1, CL_2, \dots, CL_k \rangle$, algorithm MF generates the whole confidence movement function, denoted as $F(t)$. $F(t)$ is represented as $\langle U_0(t), E_1(t), U_1(t), E_2(t), \dots, E_k(t), U_k(t) \rangle$, where $E_i(t)$ is the confidence movement function in cluster CL_i of CTP_{AMS} and $U_i(t)$ is the linkage movement function from $E_i(t)$ to $E_{i+1}(t)$. Algorithm MF (from lines 5 to 6) shows that for each cluster of CTP_{AMS} , the corresponding confidence movement functions are derived using the regression method above. However, the first time slot may not be included in CL_1 . If t_0 is the first time slot of CL_1 and $t_0 \neq 1$, the $U_0(t) = \{E_1(t_0), [1, t_0]\}$ is generated for the boundary condition. Otherwise, $U_0(t)$ will not be valid in $F(t)$. The same is true for $U_k(t)$. The linkage movement function is calculated by interpolation (in line 9 of algorithm MF).

For example, assume that $CTP_{AMS} = \langle 1, 2, 4, 5 \rangle, \langle 7, 9, 10 \rangle$, $E_1(t) = (2.333 - 2.133t + 0.867t^2 - 0.066t^3, 2.529 - 2.386t + 1.021t^2 - 0.105t^3, [1, 5])$ and $E_2(t) = (6 + 1.17t - 0.16t^2, 3 + 0t + 0t^2, [7, 10])$. It can be verified that the first time slot of cluster $\langle 1, 2, 4, 5 \rangle$ is 1. The last time slot of $\langle 1, 2, 4, 5 \rangle$ is

5 and the first time slot of cluster $\langle 7, 9, 10 \rangle$ is 7. Thus, a linkage movement function should be generated by inner interpolation. From $E_1(t)$, at the 5th time slot, we can have a data point $(x = 5.09, y = 3)$. At the 7th time slot, a data point $(x = 6.35, y = 3)$ is generated by applying $E_2(7)$. By inner interpolation, we could have $U_1(t) = (1.94 + \frac{6.35-5.09}{7-5}t, 3 + \frac{3-3}{7-5}t, (5,7))$. Similarly, $U_2(t)$ can be determined. After obtaining the confidence and linkage functions, the $F(t) = \langle E_1(t), U_1(t), E_2(t), U_2(t) \rangle$ can be derived. Figure 3.4 shows the snapshot of $F(t)$. When using $F(t)$ to predict the location of mobile users, we will only use the confidence movement function whose time interval includes the given time t . For $F(t) = \langle E_1(t), U_1(t), E_2(t), U_2(t) \rangle$, when the time is 4, only $E_1(t)$ will be used to predict the location since the given time 4 is within the time interval of $E_1(t)$.

Time Complexity Analysis: Algorithm MF is of polynomial time complexity. When the maximal size in row/column is n , the time complexity used to solve the normal equation by Strassen's algorithm is $\Theta(n^{\lg 7})$ [23]. Moreover, the interpolation by Lagrange's formula requires $\Theta(m^2)$, where m represents the number of points involved in the interpolation [23]. Since n is usually larger than m , the value of $\Theta(n^{\lg 7})$ dominates the complexity of algorithm MF.

3.2.5 Estimating A User's Location Based on a Movement Function

For many applications, it is necessary to estimate a user's location in the symbolic model. In this case, $F(t)$ represents the movement behavior of mobile users. Thus, once movement functions $F(t)$ have been obtained, the location of mobile users can be predicted as (x_t, y_t) , which denotes the coordinates of applying the movement function at time t . Through the estimated coordinate (x_t, y_t) , this coordinate can be transformed into a symbol which contains (x_t, y_t) . In our example, since each

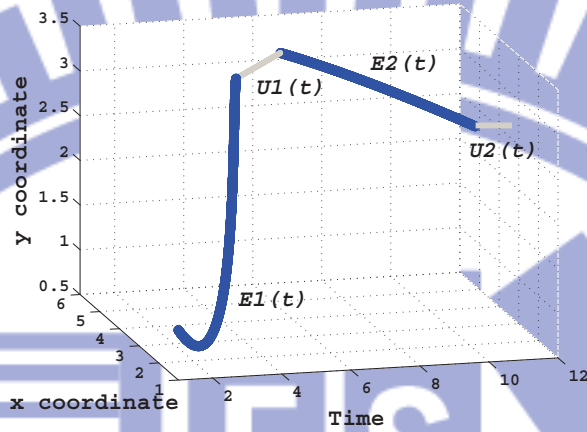


Figure 3.4: A snapshot of a complete movement function $F(t)$

base station is aware of its location and coverage area, it is easy to transform the geometric location (x_t, y_t) into the identification of the base station in the symbolic model.

3.3 Performance Evaluation

This section evaluates the effectiveness and efficiency of mining user movement patterns from call detail records. Section 4.1 presents the models for user behaviors, including movement behavior and calling behavior. Section 4.1 also describes both the synthetic dataset and the real dataset. Section 4.2 presents experimental results. Finally, the *RUMP* sensitivity analysis is given in Section 4.3.

3.3.1 Modeling User Behaviors

User behaviors in a mobile computing environment include movement behaviors and calling behaviors. This section first describes the synthetic dataset used in this study, in which user movement behaviors are derived according to pre-defined pa-

rameters. To simulate a mobile computing environment, we use a 16×16 mesh network, in which each node represents a base station. Thus, the simulation model contains 256 base stations [46]. Moreover, our simulation considers 10,000 users. As in [60], this simulation considers three movement trajectories. For each user, we randomly select one movement trajectory as his/her own movement pattern. Then, a user mostly follows his/her own movement pattern. However, users may have some movements that do not follow their movement patterns. These movements are viewed as biased movements. To prevent users from diverging too far from their movement patterns due to biased movements, we borrowed the concept in [44] that allows users to move back to their movement patterns. The number of movements made by mobile users in one time slot is modeled as a uniform distribution between $mf - 2$ and $mf + 2$. The larger the value of mf is, the more frequently mobile users move. We used the design above to generate user movements.

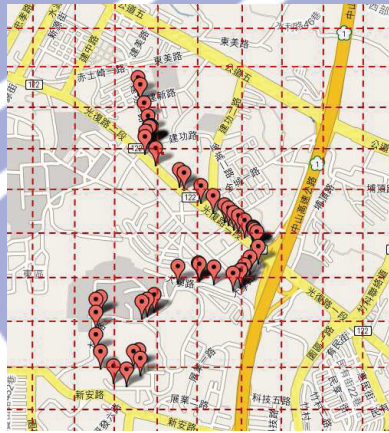


Figure 3.5: The frequent movement behavior in CarWeb dataset

However, for a real dataset, it is difficult to obtain real call detail records from mobile service providers due to the privacy issue of customers. Moreover, the *RUMP* approach requires the location information of base stations, which is

business-related information for mobile service providers. Thus, for real datasets, we use real movement logs from a GPS-based testbed, *CarWeb*, and generate simulated CDRs along with real movements. In the *CarWeb* platform, users can obtain their locations from a GPS device every five seconds and upload their locations to *CarWeb* servers. Figure 3.5 shows one frequent movement behavior, where every red flag represents a user-uploaded location. By collecting user movement behaviors for four months, we produce roughly 200 movement trajectories for each user. In the *CarWeb* dataset, the ground truth is known, which is useful to validate our mining results ⁴. In the *CarWeb* dataset, a user has frequent and infrequent movements. To simulate the coverage area of a base station, we divided the whole space into grids and viewed each grid as the coverage area of one base station. Figure 3.5 shows the grids in the *CarWeb* datasets, where the frequent movement behaviors of this user occurred within or around 16 grids. Furthermore, since the traveling times of movement sequences in the *CarWeb* dataset are not exactly the same, the traveling time for each trajectory is thus normalized to 24 hours. In both datasets, the time slot is set to 2 hours and the value of ϵ is 12.

Once user movements have been determined, calling behaviors can be modeled for each user's movements. According to [40], calling behavior can be modeled as a Poisson distribution. Moreover, a Zeta distribution is used to model burst calling behavior in this article. In a Poisson distribution, the probability that a user has x calls in a time slot is determined by $P(x) = \frac{e^{-\lambda}\lambda^x}{x!}$, where x is the number of calls and λ is the expected number of calls in a time slot. Three time slots are grouped and then each time slot is divided into three subsections, producing a total of 9 subsections in each group. For each user, the probability of having

⁴Due to customer privacy issues, it is impossible to get the ground truth of user movement behaviors even if mobile service providers were to release call detail records

phone calls in the x -th subsection of a group is $Z(x) = \frac{x^{-\lambda}}{\sum_{n=1}^{\infty} \frac{1}{n^\lambda}}$, where x indicates the subsection order in a group (i.e., the x -th subsection in a group) and λ is the value of the exponent feature for a Zeta distribution. In the beginning of subsections in a group, a user will have more phone calls, but the number of phone calls decays exponentially in the remaining subsections of a group. The speed of decay is determined by the parameter λ ; the larger this parameter is, the faster the decay is. For brevity, $CDR(\rho, \lambda)$ indicates that the calling behavior is modeled as ρ distribution with parameter λ , where the value of ρ is set to P (respectively, Z) if a Poisson (respectively, Zeta) distribution is used. For example, $CDR(P, 2)$ represents the calling behavior of a user under a Poisson distribution with $\lambda = 2$.

For comparison purposes, we also implemented the method of mining movement patterns in [46], denoted by UMP . To validate the quality of movement patterns mined by UMP and $RUMP$, we could utilize movement patterns to predict next movements of users. The accuracy of prediction indicates the quality of movement patterns mined. Hence, the hop count (referred to as hn) represents the number of base stations between the prediction location and the actual location of the mobile user. Intuitively, the smaller the value of the hop count, the closer the current location and the derived location. Thus, the expected value of hop counts per call $E(hn/call)$ is defined as $\frac{total_hop_counts}{number_of_calls}$, where the *total_hop_counts* is the sum of hop counts per call and *number_of_calls* is the total number of calls per user. To evaluate the quality of user movement patterns mined by UMP and $RUMP$, the *precision ratio* is derived and defined as $1 - \frac{E(hn/call)-1}{2n}$, where the size of network is $n \times n$ and $E(hn/call)$ is the expected value of hop counts per call. The precision ratio represents the percentage of the average hop counts from the derived cell to the current cell a mobile user with respect to the network size. Table 3.6 summarizes the definitions of some primary simulation parameters. In this table, the default values are optimal values based on following experiments

Notation	Definition	Default Value
w	Number of movement sequences	50
mf	Movement frequency	3
min_freq	Threshold used in algorithm LS	0.3
min_sim	Threshold used in algorithm LS	0.5
min_var	Threshold used in algorithm TC	0.75

Table 3.6: The parameters used in experiments

in our experimental environment. Each experimental result was obtained by an average of twenty experiments.

3.3.2 Experiments of *UMP* and *RUMP*

We first evaluated *UMP* and *RUMP* in terms of the data amount, the precision ratio, and the execution time. The data amount is the number of records stored in a movement log and a CDR log. Figure 3.6(a) shows that the data amount of *UMP* increases with the value of mf . This is because with a larger mf , a user tends to move frequently, producing a greater amount of data of the movement log. On the contrary, the data amount in *RUMP* remains almost constant. Figure 3.6(b) shows that the precision ratio of *RUMP* is smaller than that of *UMP*. However, with CDR(P, 4), the precision ratio of *RUMP* is not far below *UMP*. Note, however, that though *UMP* performs better than *RUMP* in terms of the precision ratio, it also incurs a larger amount of data in a movement log. To investigate the precision ratio gained by having the additional amount of log data, this study defines *data utilization* as the ratio between the precision ratio and the amount of log data. Figure 3.7 shows the data utilization of *UMP* and *RUMP*. With a higher mf , the data utilization of *UMP* drastically decreases. This is because the amount of data in the movement log increases dramatically as users move frequently. If the value of mf is smaller, the data utilization of *RUMP*

with a Zeta distribution is larger than that of *RUMP* with a Poisson distribution. On the other hand, when the value of mf increases, the data utilization of *RUMP* with a Poisson distribution is larger than that of *RUMP* with a Zeta distribution. It is primarily because when mf is large, it is better to have more uniform calling behaviors to allow the call detail records fully reflect user movement behaviors. These experimental results shows that *RUMP* has a higher data utilization than *UMP*. By exploring CDRs, *RUMP* is more cost-effective in mining user movement patterns.

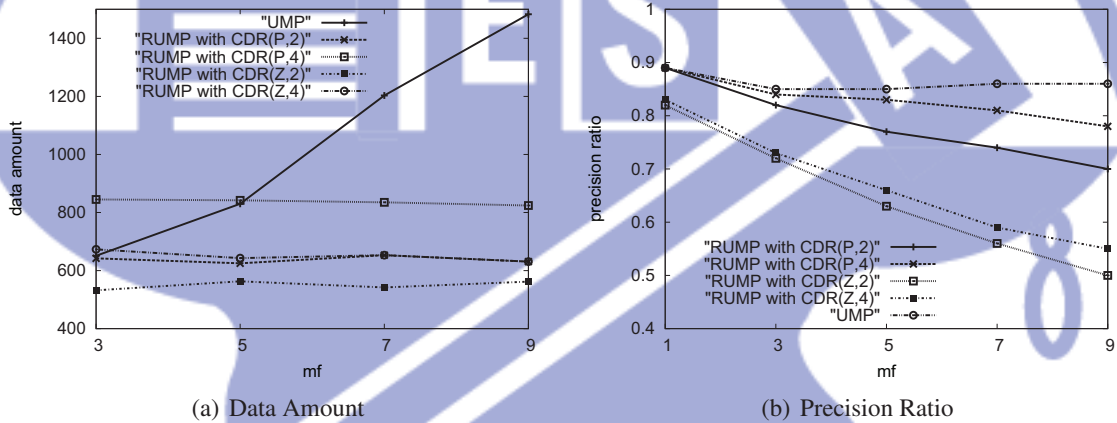


Figure 3.6: Performance comparisons of *UMP* and *RUMP* on the synthetic dataset

Figure 3.8(a) shows the data amount of *UMP* and *RUMP* with various calling behaviors under the *CarWeb* dataset. Figure 3.8(a) shows that the data amount of *RUMP* is much smaller than that of *UMP*. Furthermore, Figure 3.8(b) shows that the precision ratios of *UMP* and *RUMP*, indicating that the difference between *UMP* and *RUMP* is not large. This suggests that *RUMP* is able to achieve acceptable precision ratios when using a smaller amount of data. However, through performing better than *RUMP* in terms of the precision ratio, *UMP* incurs more amount of data in the movement log. In Figure 3.9, the

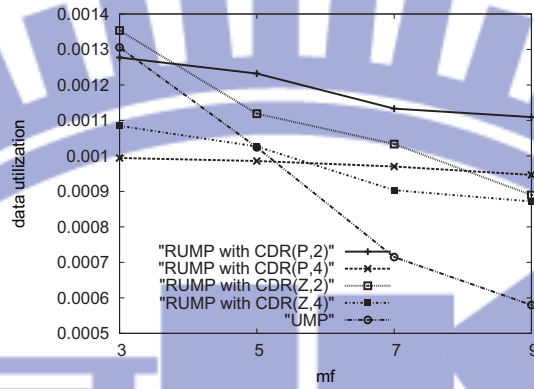


Figure 3.7: Data utilization of *UMP* and *RUMP* on the synthetic dataset

data utilization of *UMP* is much smaller than that of *RUMP*, showing that with a smaller amount of log data, *RUMP* can still achieve an acceptable precision ratio.

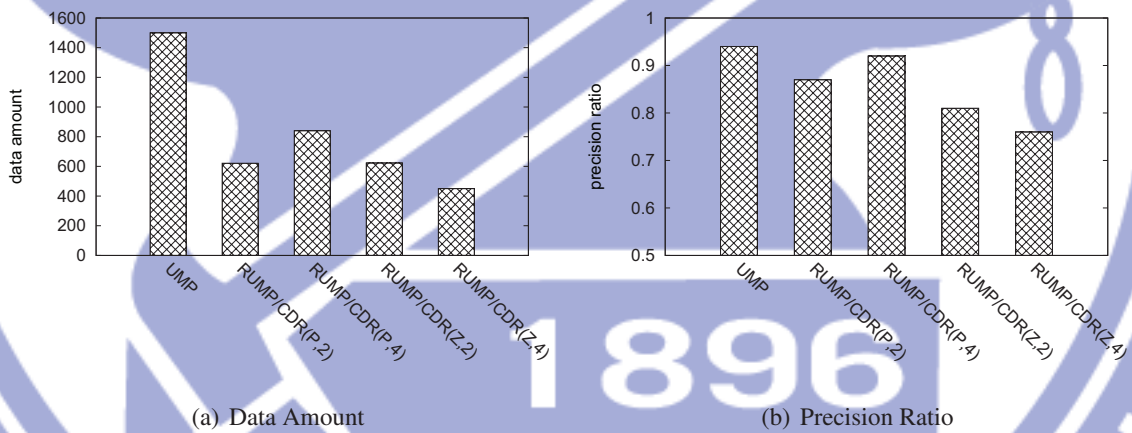


Figure 3.8: Performance comparisons of *UMP* and *RUMP* on the *CarWeb* dataset

Figure 3.10 shows the execution time of *UMP* and *RUMP* under the synthetic dataset. Figure 3.10(a) shows that the *RUMP* execution time is smaller than that of *UMP* in both the synthetic dataset and the *CarWeb* dataset. With a larger number of movement sequences, the *UMP* execution time significantly in-

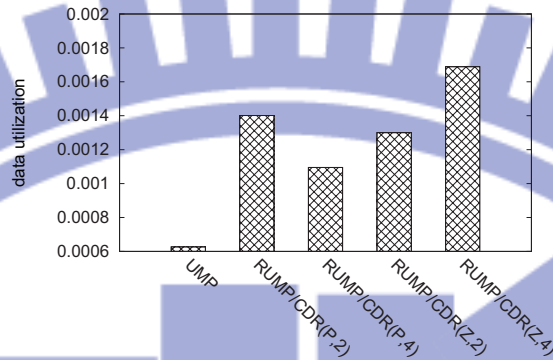


Figure 3.9: Data utilization of *UMP* and *RUMP* on the *CarWeb* dataset

increases. With a higher mf , the execution time of *RUMP* becomes much slower than that of *UMP*. Further, *RUMP* has better scalability than *UMP*. In addition, Figure 3.10(b) shows the execution time of *UMP* and *RUMP* on the *CarWeb* dataset. Similar to the results in the synthetic dataset, the *RUMP* execution time is much smaller than that of *UMP*. As the number of movement sequences increases, *UMP* takes longer to discover user movement patterns. On the other hand, the *RUMP* performance is determined by the data amount generated by calling. Since the data amount generated by calling is usually fewer than that by movements, *RUMP* incurs a smaller execution time.

3.3.3 Sensitivity Analysis of *RUMP*

This section further investigates the parameters used in *RUMP*. First, the impact of w is presented. Then, we examine the impact of thresholds on the mining results.

Impact of w

Figure 3.11 shows the experiments of varying w values for *RUMP* under both the synthetic dataset and the *CarWeb* dataset. This figure indicates that the *RUMP*

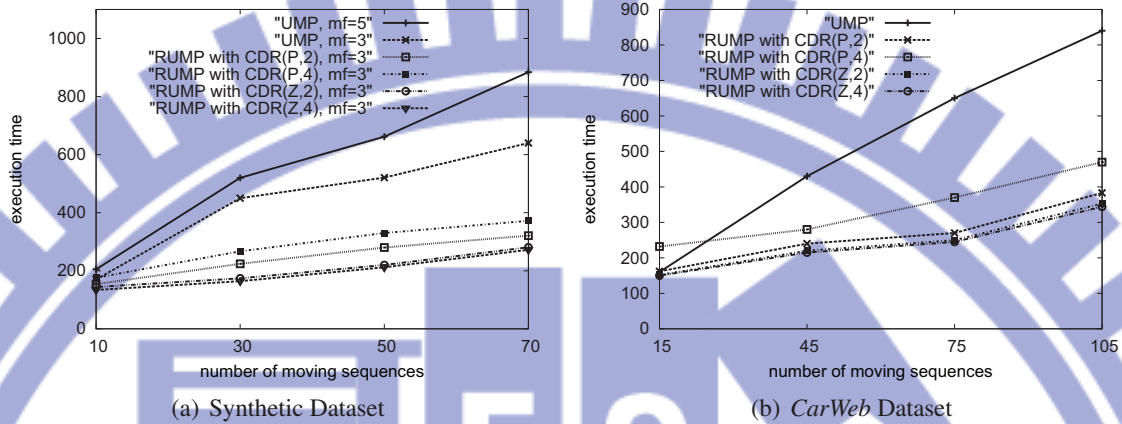


Figure 3.10: Execution time for various numbers of movement sequences

precision ratio increases as the value of w increases in both datasets. This is because as the value of w increases, the number of movement sequences considered in *RUMP* increases as the value of w increases. In this case, *RUMP* can use more calls to discover user movement patterns. The *RUMP* precision ratio with a Poisson distribution is larger than that of *RUMP* with a Zeta distribution. This is because the calling behavior in a Poisson distribution is much more evenly across user movements. Thus, *RUMP* is able to fully capture user movement behaviors when the calling behavior follows a Poisson distribution. In a Poisson distribution, with a larger value of λ , the precision ratio of *RUMP* is larger. For a larger value of λ , the amount of call detail records tends to increase, thereby reflecting the complete movement behaviors of users. For users with a larger number of calls and non-burst calling behavior, the value of w can be set smaller to quickly obtain movement patterns. In contrast, for users with a smaller number of calls or burst calling behavior, the value of w should be set larger to improve the precision ratio of the movement patterns mined by *RUMP*.

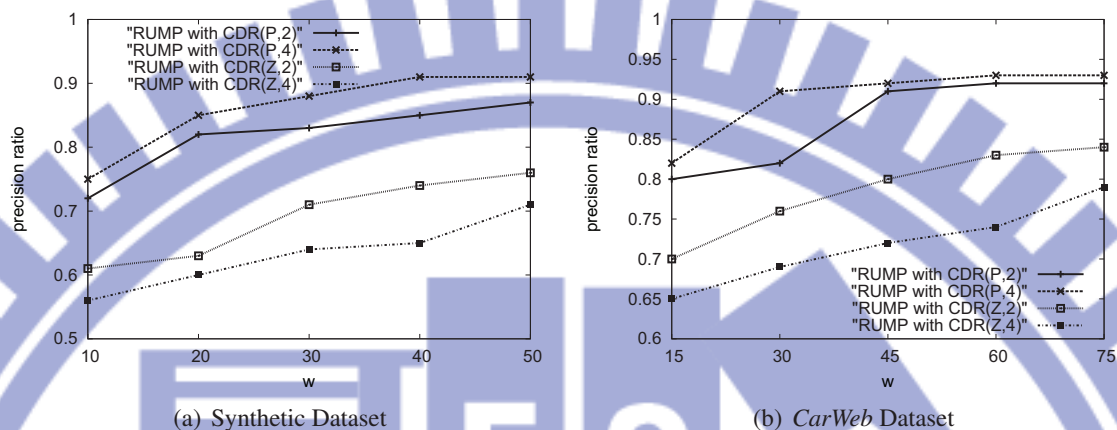


Figure 3.11: Precision ratio of *RUMP* with varying w

Impact of Thresholds in Algorithm LS

This section examines the impact of min_freq and min_sim on the *RUMP* performance. Algorithm LS uses min_freq and min_sim thresholds to extract CDRs representing frequent movement behaviors. Figure 3.12 and Figure 3.13 show *RUMP* experiments with varying values of min_freq and min_sim . Figure 3.12(a) shows the result of using *RUMP* on the synthetic dataset. This figure indicates that the *RUMP* precision ratio tends to increase when the value of min_freq increases from 0.1 to 0.3. This figure also shows that the *RUMP* precision ratio decreases when min_freq exceeds 0.3. This is because increasing min_freq filters out areas through which users do not frequently move are filtered out. However, a larger min_freq is too strict for identifying what areas are frequent and decreases the precision ratio. Figure 3.12(b) shows that the same phenomenon for the *CarWeb* dataset. Selecting the value of min_freq should be determined empirically. For example, in this experiment, we set min_freq at 0.3.

Figure 3.13 shows the *RUMP* precision ratio with various values of min_sim . In both datasets, the *RUMP* precision ratio tends to increase when min_sim in-

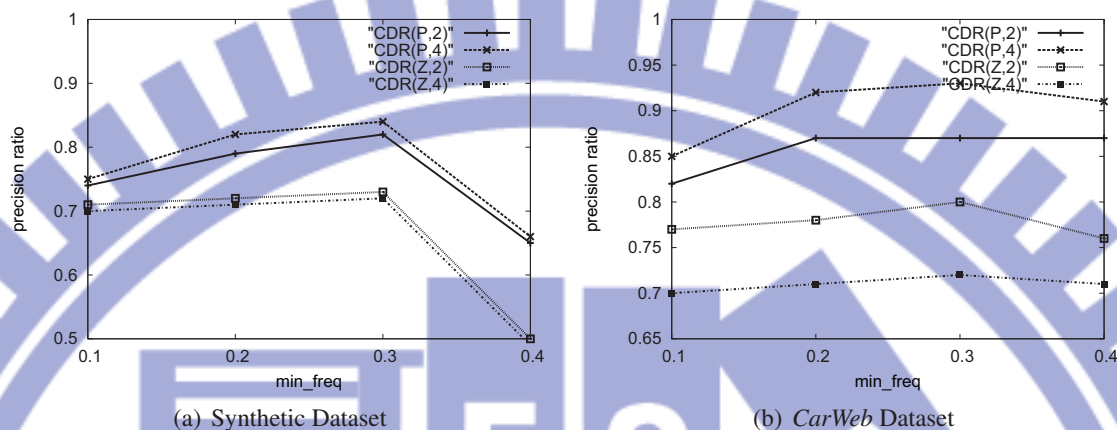


Figure 3.12: Precision ratio of *RUMP* with min_freq varied

increases from 0.1 to 0.5. However, when the value of min_sim exceeds than 0.5, the *RUMP* precision ratio decreases. The min_sim threshold is set to identify whether or not a movement sequence is similar to the frequent movement behavior. With a larger value of min_sim , only a few movement sequences are identified as being similar to frequent user movement behaviors. This, in this turn, decreases the *RUMP* precision ratio. Therefore, the value of min_sim should be carefully set. Experimental results shows that min_freq should be set to 0.3 and min_sim should be set to be 0.5 to achieve the best precision ratio performance.

Impact of Thresholds in Algorithm TC

As described above, the value of min_var for algorithm TC affects the accuracy of the *RUMP* time clustering results. We conducted experiments to examine the impact of min_var . For the synthetic dataset, Figure 3.14(a) shows that the precision ratio of *RUMP* with the values of threshold min_var varied. This figure indicates the *RUMP* precision ratio significantly increases when min_var is 0.25. However, the precision ratio of *RUMP* decreases when min_var exceeds than 0.75. This is because excessively large values of min_var result in most of

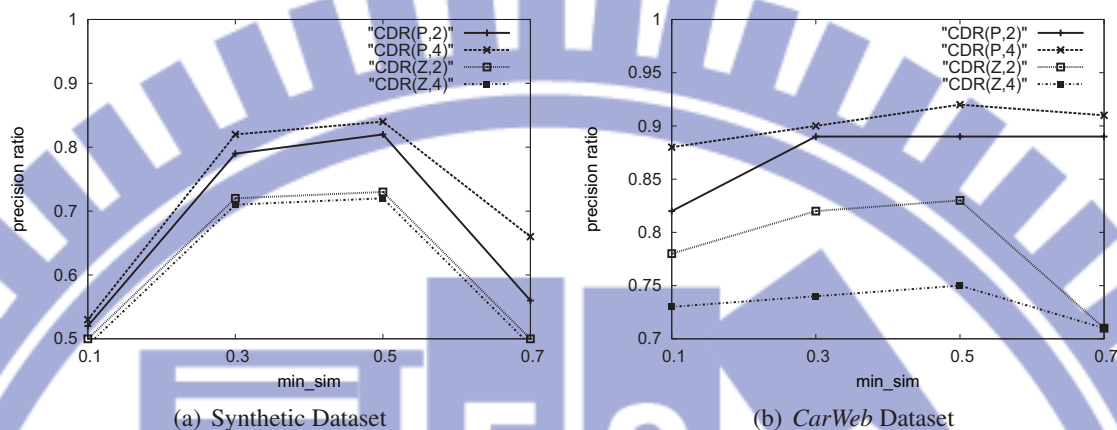


Figure 3.13: Precision ratio of *RUMP* with min_sim varied

the call detail records being grouped in the same cluster. Hence, the number of movement functions is not enough to capture user movement behaviors. Furthermore, with a larger mf , the *RUMP* precision ratio is smaller and significantly decreases when min_var is larger. For the *CarWeb* dataset, Figure 3.14(b) shows the similar experimental results. These results indicate that min_var should be set to be a smaller value for users who move frequently. The value of min_var , which can be determined empirically, should not set too larger. For example, in Figure 3.14(a), min_var should set to 0.75 because the *RUMP* precision ratio is the highest.

Figure 3.15 depicts the *RUMP* precision ratio with various calling behaviors. In Figure 3.15(a), the results of CDR(P,2) and CDR(P,4) are similar to the results above. However, it is interesting to note that the precision ratios of CDR(Z,2) and CDR(Z,4) do not decrease when the value of min_var exceeds than 0.75. Since burst calls happen in the beginning of every three time slots, most of the call detail records in these three time slots can be grouped into one cluster. Figure 3.15(b) shows the similar results in the *CarWeb* dataset. Thus, we can set min_var as 0.75 to obtain the highest *RUMP* precision ratio.

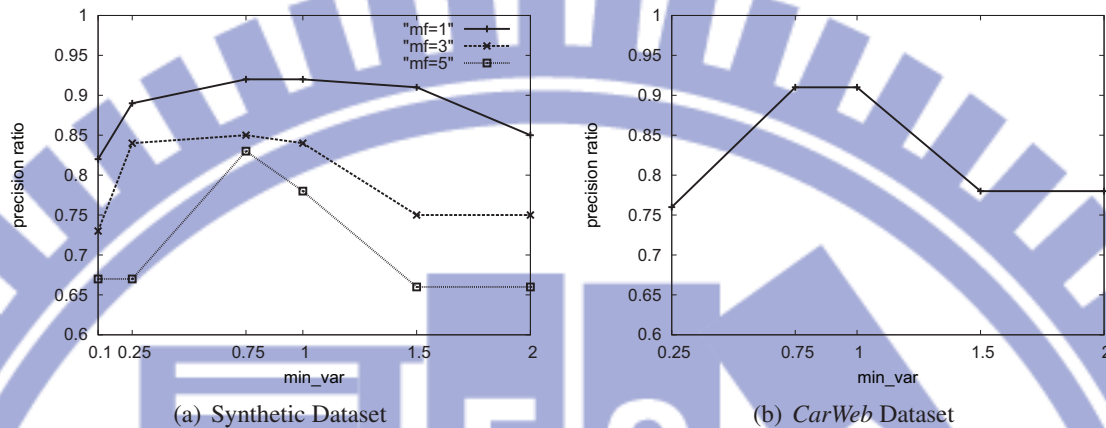


Figure 3.14: Precision ratio with min_var varied

3.4 Conclusions

User movement patterns can provide a lot of benefits in many mobile design schemes and applications, including designing a paging area, developing data allocation schemes, conducting querying strategies, or offering navigation services. This article proposes a regression-based approach called *RUMP* for mining user movement patterns from call detail records. To fully exploit the fragmented spatio-temporal information hidden in such trajectories, the proposed regression-based solution discovers user movement patterns. The *RUMP* approach uses three algorithms. First, algorithm LS extracts CDRs that reflect the frequent movement behaviors of mobile users. By capturing similar movement sequences from call detail records, an aggregation movement sequence is computed to represent the frequent movement behaviors of mobile users in each time slot. The feature of spatio-temporal locality states that if the time interval between consecutive calls is small, the mobile user is likely to have moved nearby. By exploring this feature, algorithm TC is able to determine the number of regression functions properly by clustering those movement records whose time of occurrence are very close from

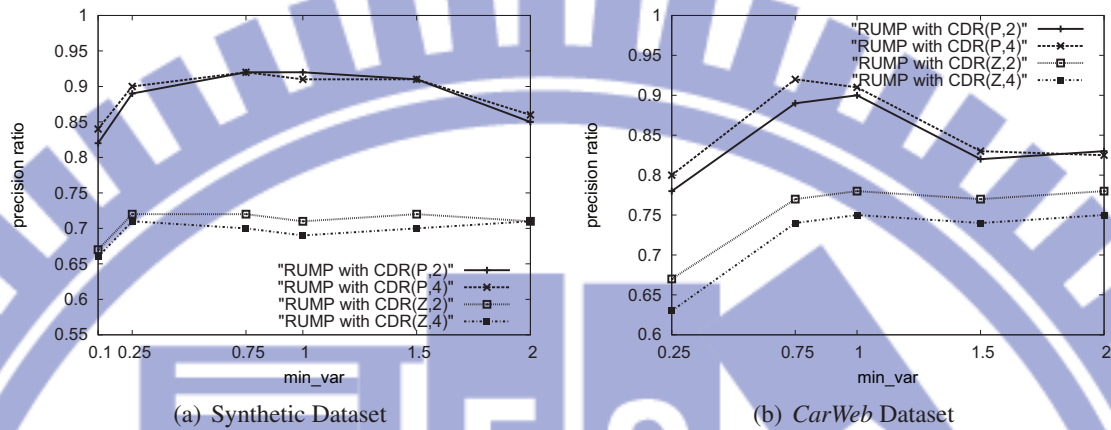


Figure 3.15: Precision ratio of *RUMP* with min_sim varied

an aggregation movement sequence. For each cluster of the aggregation movement sequence, algorithm MF generates the movement functions representing user movement patterns of mobile users. This article evaluates the performance of the proposed algorithms and conducts sensitivity analysis on several design parameters. Experimental results indicate that *RUMP* can efficiently and effectively derive user movement patterns that capture the frequent movement behaviors of mobile users.

Chapter 4

CACT: Clustering and Aggregating Clues of Trajectories for Mining Trajectory Patterns

4.1 Introduction

With the pervasiveness of mobile devices nowadays, the location of users can be easily determined using GPS devices and positioning techniques. Using smart phones, people can access location-based services and share their locations with friends via social web sites, such as Google's latitude service and Foursquare [1][2][3]. As a result, an increasing amount of user trajectory data has become available. Given a set of users' trajectories, mining trajectory patterns has many potential applications, such as urban planning and hot spot detection. In particular, given individual user trajectories, one could discover individual trajectory patterns, which are particular useful for the design of personalized navigation services or for location-based recommending services. In this paper, we aim to discover individual trajectory patterns.

A considerable amount of research efforts has elaborated on mining trajectory patterns [5][18][44][30]. In general, trajectory patterns are sequences of regions that a user usually appears. One of the challenges in trajectory pattern mining is

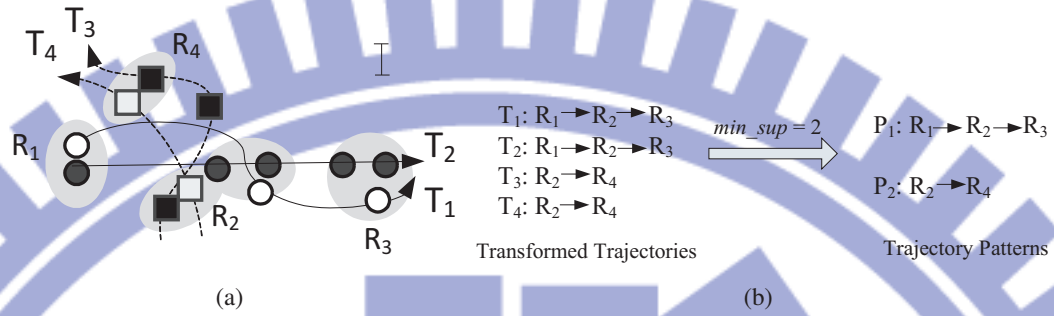


Figure 4.1: An example of trajectory pattern mining.

how to define a hot region, a basic unit of trajectory patterns. Hot regions refer to the range where a user frequently appears. Prior research has explored the density concept, which identifies hot regions as those that contain a sufficient number of data points [44][30]. These hot regions are the basic units in trajectory patterns. Given a set of trajectories, prior works in [44][30] have directly employed density-based clustering algorithms (e.g., DBSCAN) to identify hot regions. A variety of definitions for density measurements have been proposed. In [18], the whole space is divided into grids. The density in a grid is defined as the number of trajectories that cross this grid. Grids that have higher densities form a compact region and are regarded as a hot region. The number of grids substantially influences the determination of hot regions and is difficult to determine. The authors in [5] transform trajectory data into a series of line segments. If several line segments from different trajectories are close, a hot region that contains these line segments is thus determined. Clearly, without proper determining hot regions, trajectory patterns cannot truly reflect movement behaviors of users.

Once hot regions are determined, they can be viewed as a feature for describing user movement behaviors. Trajectories that pass through some common hot regions usually indicate the same movement behavior¹. Prior works transform

¹Movement behaviors could be defined as a set of moving paths sharing common movement

trajectories into sequences of hot regions. Given a set of hot region sequences, trajectory pattern mining algorithms have been proposed to discover sequential relationships among hot regions. Explicitly, trajectory patterns could be viewed as frequent movement behaviors of a user. In reality, a user may have multiple movement behaviors hidden in his trajectories. Without clustering trajectories, hot regions determined by prior works could not accurately capture user movement behaviors. Assume that one user has two movement behaviors, as shown in Figure 4.1(a), where T_1 and T_2 represent one movement behavior, whereas T_3 and T_4 refer another movement behavior. Identical to [44][30], DBSCAN are used to extract hot regions from these four trajectories. As can be seen in Figure 4.1(a), five hot regions are derived. According to these hot regions, four trajectories are represented as sequences of hot regions. Assume that the frequent threshold for mining trajectory patterns is 2 (i.e., a sequence of hot regions is identified as a trajectory pattern if this sequence of hot regions appears in at least 2 times among a given set of hot region sequences). Then, we could derive two trajectory patterns (i.e., $R_1 \rightarrow R_2 \rightarrow R_3$ and $R_2 \rightarrow R_4$). Clearly, trajectory pattern $R_1 \rightarrow R_2 \rightarrow R_3$ (respectively, $R_2 \rightarrow R_4$) demonstrates a movement behavior captured by trajectories T_1 and T_2 (respectively, T_3 and T_4). Note that hot region R_2 contains some location points of four trajectories that capture different kinds of movement behaviors. In this case, R_2 cannot precisely identify movement areas when a user stays at R_2 . R_2 implies the area in which the user is likely to stay. In fact, this user never stays in the left part of area R_2 if the user follows the movement behavior captured by T_1 and T_2 . Consequently, in this paper, we argue that to derive trajectory patterns, trajectories should be first judiciously clustered into several groups, where each group is likely to represent one movement behavior. Then, trajectories features such as velocity, acceleration, and so on [15]. In this paper, we study the movement behavior in terms of hot regions.



Figure 4.2: An example of trajectories from CarWeb dataset.

in each group are used to derive hot regions and these hot regions are able to truly reflect movement areas where a user usually appears.

For trajectory clustering, one should formulate a similarity measurement among trajectories according to trajectory characteristics. Generally speaking, a trajectory consists of data points with locations and the corresponding occurrence time. As has been reported in many prior works, trajectories may exhibit certain spatial and temporal shiftings, which indicate that the locations and occurrence time of data points in two trajectories are not usually the same even if two trajectories capture the same movement. For example, consider two trajectories T_1 and T_2 in Figure 4.2(a) and Figure 4.2(b), where the roman number associated with each data point represents the generation order of data points and the time of data points

is represented as one time slot. For each data point of T_1 and T_2 , data points are not exactly the same in terms of spatial and temporal domain even if a user follows the same movement behavior. Since positioning devices usually have spatial bias, the spatial information of data points with the same order is not always the same, which shows the spatial shifting. Moreover, as can be seen in Figure 4.2(d), T_2 is delayed for approximately 2 time slots compared to T_1 , which demonstrates the temporal shifting. However, prior works have rarely considered an important characteristic of trajectory data: *silent duration*. The *silent duration* represents a time duration when there is no data point available. Due to silent durations, trajectories could not record detailed movements of users. The existence of silent durations comes from the fact that trajectories are generated by positioning devices. Since a user's movement is continuous in both the time and spatial domain, obtaining trajectory data could be viewed as sampling from his movement². Thus, trajectories only capture user movements in a discrete manner. Moreover, in reality, data points may not successfully be sampled since environmental factors (i.e., weather conditions or high building) have a considerable impact on the positioning of user locations. Due to the disturbance of environmental factors, although data points of trajectories are collected by every fixed time interval, trajectory data usually still has some missing data points. For some real applications, trajectory data are collected by smart phones via WiFi or 3G networks. If smart phones are in the heavy loads, data points of trajectories cannot be successfully transmitted. Thus, for the reasons above, trajectory data points are not ideally collected such that silent durations exist in trajectory data. Figure 4.2(c) shows an example of silent duration, where T_3 captures the same movement behavior recorded by T_1 . From Figure 4.2(d), data points of T_3 at time slots $t = 2, 4, 5$ are not collected.

²In the rest of this paper, the terms *movement* and *trajectory* have different meanings: a movement refers to the actual path how a user moves; a trajectory means a sequence of locations sampled by the positioning device from a movement.

Therefore, these two time durations are silent durations.

Note silent durations are affected by sampling methods as well. Currently, two types of sampling methods are provided: one is to sample user movements every fixed time interval (referring to as sampling-by-time) and the other is to sample user consecutive movements at every fixed geographical interval (referring to as sampling-by-distance). Using various sampling strategies, silent durations may distribute in the different way. Figure 4.3 shows two trajectories generated by the sampling-by-distance and the sampling-by-time strategies, respectively. In Figure 4.3(a), although the data points in T_4 are evenly distributed in the spatial space, silent durations exist in T_4 . Similarly, Figure 4.3(b) shows that T_5 derived by sampling-by-time still has silent durations. Regardless of which sampling methods are used, silent durations exist in trajectories. A considerable amount of research efforts has been elaborated on the formulation of similarity measurements to deal with spatial/temporal bias, noise and spatial/temporal shifting. However, with silent durations, trajectories that capture the same movement behaviors will have different number of data points. Consequently, this paper develops a new kind of similarity measurement. This paper also discusses and compares existing similarity measurements, such as Euclidean distance, dynamic time warping [35] and edit-distance-based approaches [8], and our proposed one are presented latter.

In this paper, we propose a framework CACT (standing for Clustering and Aggregating Clues of Trajectories) for discovering trajectory patterns. Due to silent durations, even capturing the same movement behavior, trajectories may only reflect fragmented movement information about this behavior. However, these fragmented trajectories may provide some *clues* to indicate whether they represent the same movement behavior or not. An intuition is that two trajectories are likely to be from the same movement behavior if they have a lot of data points that are close in the spatial and temporal domains. These data points are referred as clues



Figure 4.3: Two sampling methods for collecting trajectories.

between these two trajectories. With the observation of clues, a similarity measurement CATS (standing for *Clue-Aware Trajectory Similarity*) is proposed in that clues are identified and considered. In light of CATS, a clustering algorithm CATC (standing for *Clue-Aware Trajectory Clustering*) is proposed to fully exploit clues between trajectories to group trajectories into several clusters such that each cluster represents one movement behavior of a user. For each cluster, an aggregation algorithm CATA (standing for *Clue-Aware Trajectory Aggregation*) is proposed to aggregate trajectories as a sequence of hot regions. To the best of our knowledge, this is the first work to deal with clues of trajectories, let alone proposing a series of clue-aware algorithms for trajectory pattern mining. Extensive experiments on both real and synthetic datasets are conducted. We compare our proposed algorithms with existing similarity measurements and clustering algorithms. The results show that CACT can discover trajectory patterns effectively even trajectories do not contain the complete movements of a user.

The rest of the paper is organized as follows. Related works are discussed in Section 4.2. Preliminaries are given in Section 4.3. Our proposed framework

CACT and the corresponding algorithms are presented from Section 4.4 to Section 4.6. Experimental results are shown in Section 4.7. Finally, Section 4.8 concludes with this paper.

4.2 Related Works

In this section, we will first present some existing similarity measurements for trajectory data. Then, a survey for trajectory clustering is discussed. Finally, related works about trajectory pattern mining are described.

4.2.1 Similarity Measurements

As mentioned before, for clustering trajectory data, one important task is to formulate a similarity measurement between two trajectory data. A considerable amount of research efforts have been elaborated on similarity measurements on both trajectory data such as LCSS [57], ERP [7], EDR [8], and RTSD [54] and time series data, including dynamic time wrapping [62][35], SpADe [9], and wDF[13].

Since our goal is to identify similar movement behaviors from trajectories, we will compare our proposed similarity with LCSS, DTW, EDR and wDF. Table 4.1 summarizes these existing similarity measurements and their capabilities. To evaluate their capabilities, there are five criteria. The first criterion is the capability to handle the *local temporal shifting*. The second one is how to deal with *time sensitive* which represents the ability to distinguish whether two trajectories are from different time periods or not. For example, two trajectories with the same shape should be distinguishable if one trajectory occurs in the morning and the other trajectory happens in the evening. Since data points have spatial bias, the third criterion, *space quantization*, is how to assign weights to overcome spatial bias. According to the distance between two data points from different trajectories, previous similarity measurements assign different weights for data points

Functions	Local Temporal Shifting	Time Sensitive	Space Quantization	Mapping Scheme	Empty Mapping
DTW	√		none	1-n/n-1	
LCSS	√	√	discrete	1-1	√
EDR	√		discrete	1-1	√
wDF	√	√	none	1-1	

Table 4.1: Comparison of similarity measurements.

and the weights will be aggregated as a score for two trajectories. There are two kinds of space quantization: one is the discrete scheme and the other is the continuous scheme. For the discrete scheme, if the distance between two data points is larger than a spatial threshold, the weight for these two data points is set to 0. Otherwise, the weight is set to 1. For the continuous scheme, data points whose distance is smaller than the spatial threshold will further assign different weights. Given two trajectories (e.g., T_i and T_j), a *mapping scheme* represents how to map one data point of trajectory T_i to data points of trajectory T_j . Mapping scheme 1-n denotes that a data point of T_i is able to map n data points of T_j , scheme n-1 denotes that n data points of T_i could map to one data point of T_j , and scheme 1-1 presents that one data point of T_i can only map to only one data point of T_j . For example, DTW has two mapping schemes n-1 and 1-n. On the other hand, the mapping scheme of LCSS is 1-1 because a point is only mapped to the other one point. *Empty Mapping* represents that if a data point of one trajectory cannot find a suitable data point for mapping, one could skip the mapping for this data point.

Note that RTSD and SpADe are not suitable for trajectory pattern mining in this paper. Explicitly, RTSD considers two trajectories to be similar if several translations and rotations are performed to two trajectories. Thus, a hurricane that has its movement from South to South-East is recognized to be similar to another hurricane that has its movement from West to South-West. However, this property is not proper for trajectory pattern mining in this paper since trajectories with simi-

lar movement directions are clustered, which shows that rotations are not allowed. As for SpADe, SpADe focuses on the shape-based pattern detection for time series. The main goal of SpADe is to use "shape" to distinguish whether two time series are similar or not. SpADe first finds local patterns and then calculates the distance of two time series by calculating the shortest path between these matched local patterns. To recognize whether two trajectories have the same shape, SpADe tolerates certain amplitude-scaling and amplitude-shifting. However, this may be not desired in mining trajectory patterns: two trajectories with the same shape could not represent the same movement behavior if they are not close enough in both the spatial and temporal domains. Moreover, due to silent durations, local patterns may not always be recognized since there is no information available in silent durations. Thus, we only consider LCSS, DTW, EDR and wDF for performance comparisons. Since no exiting similarity measurement is developed for handling silent durations of trajectories, our proposed similarity measurement is thus different from others.

4.2.2 Trajectory Clustering

The related work of trajectory clustering could be categorized into two kinds: one is to cluster the entire trajectories and the other is to cluster sub-trajectories.

For the first kind of trajectory clustering, a model-based approach for trajectory clustering is proposed [16]. The authors in [16] assume that each trajectory is smooth such that each data point could be estimated by a probability density function. Given density probability functions that a trajectory belongs to clusters, the probability of each observed trajectory could be modeled as a mixture density function. Therefore, this algorithm first represents a set of trajectories by a regression mixture model, and then uses maximum-likelihood principle to determine the cluster memberships. However, the major limitation is that the probability density

function of each cluster and the number of clusters should be given. In [45], the authors proposed an adaptation of a density-based clustering algorithm to trajectory data based on a simple notion of distance between trajectories. This work proposed a distance function that first computes the Euclidean distance between locations at each time slot and then averages the sum of Euclidean distance values at all time slots. Based on this function, a particular density-based clustering algorithm, OPTICS is used to cluster trajectories. Moreover, an empirical comparison with several traditional k-means and hierarchical algorithms showed that OPTICS is the most suitable clustering algorithm for clustering trajectories. The advantage from the previous one is that the number of cluster does not need to be known in advance. However, data points at every time slot are required to be available (or be well-approximated) for the computation of the proposed distance function. Thus, this approach may not work very well when there are silent durations in trajectories.

For the second kind of trajectory clustering, the state-of-art approach is TraClus [38]. The primary goal of this work is to discover common sub-trajectories from a set of trajectories. Different from the first class, the key observation of this work is that clustering trajectories as a whole could miss common sub-trajectories. Discovering common sub-trajectories could be very useful in many applications, especially if we have regions of special interest for analysis. Therefore, the authors in [38] proposed a partition-and-group framework for clustering sub-trajectories. This framework first decomposes a trajectory into a set of line segments, and then groups similar line segments together into a cluster. In our work, we need to cluster the whole trajectories instead of sub-trajectories. In addition, we slightly utilize TraClus to find common sub-trajectories among trajectories and then formulate the similarity of trajectories using these common sub-trajectories. We will compare our proposed clustering algorithm with a revised TraClus later. The above

trajectory clustering algorithms work well under the assumption that trajectories could fully represent the movements of objects or could be well-approximated.

4.2.3 Trajectory Pattern Mining

The problem of mining trajectory patterns has attracted a considerable amount of research efforts. Generally speaking, the flow of mining trajectory patterns is to find hot regions and then derive the sequential relationship among hot regions. On discussing the form of trajectory patterns, the authors in [44][24][56] have elaborated on mining spatio-temporal association rules of hot regions from a set of trajectories. In [5], the authors claimed the fuzziness of locations in patterns and developed algorithms to discover spatio-temporal sequential patterns. Furthermore, the authors in [33] proposed a clustering-based approach to discover moving regions within time intervals. The authors in [38][39] proposed a framework to cluster sub-trajectories which first partition trajectories into small segments and cluster them according to their geometric properties (i.e, angle, distance, length). In [30], the authors developed a hybrid prediction model, consisting of vector-based and pattern-based model, to predict movements of users. In [18] and [17], the authors exploited temporal annotated sequences in which sequences are associated with time information (i.e., transition times between two movements).

We mention in passing that some works deal with a trajectory convoy problem and a moving clustering problem. In the trajectory convey problem [31], a convoy represents a group of objects that travel together for more than some minimum time duration. For example, in Figure 4.4(a), given the minimum time duration as 3, these five objects belong to one convoy because they travel together for sufficient time duration. Clearly, our trajectory patterns are different from the trajectory convey in that a trajectory pattern indicates sequential relationships among

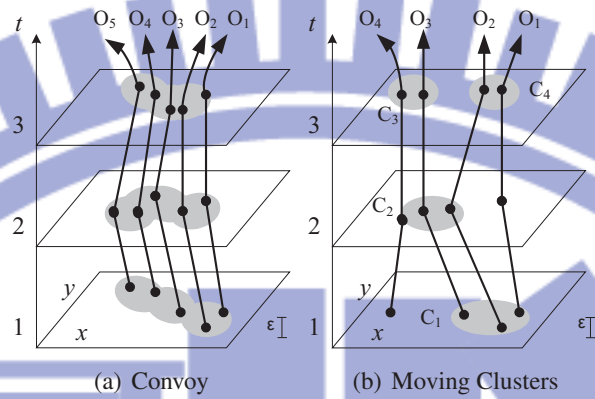


Figure 4.4: Examples for trajectory convoy and moving clusters.

hot regions, whereas a convoy indicates a set of objects that have spatial relationships at some time slots. Clearly, the outputs of these two problems are significantly different. Moreover, the distance between objects in a cluster at a time slot may be larger. For example, the distance between O_1 and O_5 is large, reflecting the same problem of hot regions determination by density-based approaches. If trajectories have silent durations, the trajectory covey will not discover good clusters at each time slot. As for the moving clustering problem, the goal is to derive a sequence of spatial clusters to a sequence of spatial clusters at consecutive time slots such that the intersection of spatial clusters contains a sufficient number of objects. In other words, the number of objects in the intersection of spatial clusters should be larger than a threshold [33]. For example, in Figure 4.4(b), if the threshold for the portion of common objects is $1/2$, C_1 and C_2 become a moving cluster. Notice that moving clusters refer to the spatial group of objects over time. However, our trajectory pattern mining is to discover frequent movement behavior. The moving clustering problem only considers the number of common objects among spatial clusters and objects in the set of moving clusters are not always the same. Hence, moving clusters cannot directly reflect frequent trajectory patterns. Both the trajectory convoy and moving clusters work well with detailed

trajectories. The most challenge point in our paper is that trajectories may not have detailed movements and thus have some silent durations. previous works do not cluster trajectories before the determination of hot regions. Furthermore, for trajectory clustering, the issues to deal trajectories which can only reflect partial movement behaviors are not addressed. By fully exploiting trajectory clues, our proposed approach is able not only identify the various movement behaviors but also accurately discover trajectory patterns. These features distinguish our works from others.

4.3 Problem of Trajectory Pattern Mining

A trajectory T_i of a user is defined as a time-order sequence of data points and T_i is typically expressed by $T_i = \langle p_{i,1}, p_{i,2}, \dots, p_{i,n} \rangle$, where $p_{i,j} = (\ell_{i,j}, t_{i,j})$ represents the location of the user (i.e., $\ell_{i,j}$) at the time $t_{i,j}$, $t_{i,j} < t_{i,j+1}$ and n is the length of trajectory T_i . The location $\ell_{i,j}$ is usually a two-dimensional or three-dimensional data point. Given a set of trajectories, we intend to mine sequential relationships among hot regions. In this paper, we extend the definition of hot regions in [5]. Explicitly, the hot region contains not only the spatial area but also temporal time interval to indicate where and when a user appears. The definition of a *hot region*, a basic unit of trajectory patterns, is defined as follows:

Definition. Hot Region: Given a spatial threshold ϵ and a temporal threshold τ , a hot region is a spatial-temporal prism structure r_i that satisfies two criteria: (1). In the temporal domain, hot region r_i has its time interval, denoted as $[r_i.S, r_i.E]$, where $r_i.S$ is the start (respectively, end) time of r_i . The time duration $r_i.E - r_i.S + 1$ should be larger than τ . (2). In the spatial domain, the projection of r_i to the spatial domain, represented as two dimensional XY-plane,

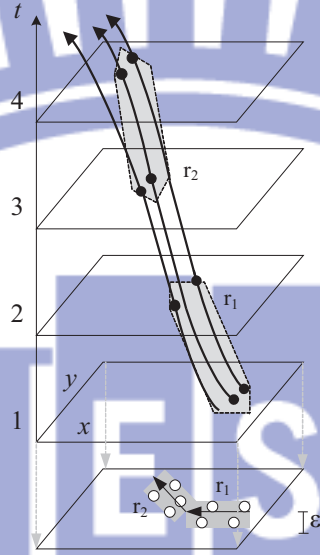


Figure 4.5: A trajectory pattern.

is a rectangle, in which there is a representative line \vec{L}_i such that for each data point x in this rectangle, the distance between data point x and the \vec{L}_i should be smaller than ϵ .

For example, three trajectories are shown in Figure 4.5, where data points of trajectories are marked as black points at the corresponding time slots. Given $\tau = 2$ and ϵ , two hot regions r_1 and r_2 , represented as two spatial-temporal prisms, are generated. These two hot regions r_1 and r_2 have their time duration as 2 and the spatial projections of r_1 and r_2 are rectangles with their representative lines.

With the definition of hot regions above, a trajectory pattern TP is represented as an ordered sequence of hot regions $TP=r_1r_2\dots r_k$, where k is the length of TP . Each trajectory pattern TP is derived from a set of supporting trajectories. The definition of supporting trajectories for trajectory pattern TP is as follows:

Definition. Supporting Trajectory: Given a trajectory pattern $TP = r_1r_2\dots r_k$, for each hot region r_i of TP , a supporting trajectory with respect to TP has some data points, e.g., p , that the occurrence time of p is within $r_i.S$ and $r_i.E$, and $d(p, \vec{L}_i) < \epsilon$, where $d(\cdot)$ is the geographic distance function.

In this paper, given a set of trajectories and three thresholds, min_sup , ϵ , τ and λ , the problem is to discover trajectory patterns which have more than min_sup supporting trajectories and those supporting trajectories should have their clue similarity values larger or equal to λ . Clearly, a trajectory pattern refers a frequent movement behavior since each trajectory pattern has at least min_sup supporting trajectories.

As pointed out early, a user may have multiple movement behaviors. As such, we claim that before determination of hot regions, we should perform trajectory clustering to derive a set of clusters that represent different movement behaviors. Each cluster represents one trajectory pattern and then, for each cluster, we derive hot regions of each trajectory pattern. Our proposed framework CACT consists a series of algorithms, including clue-aware trajectory similarity (CATS), clue-aware trajectory clustering (CATC) and clue-aware trajectory aggregation (CATA). The overview of our proposed framework CACT is shown in Figure 4.6. In the following sections, we will detail each algorithm in our framework CACT.

4.4 Clue-Aware Trajectory Similarity

In this section, we first discuss the unique characteristics of trajectories and then, our design of similarity measurement is presented. Finally, some interesting properties of clue-based similarity measurements are described.

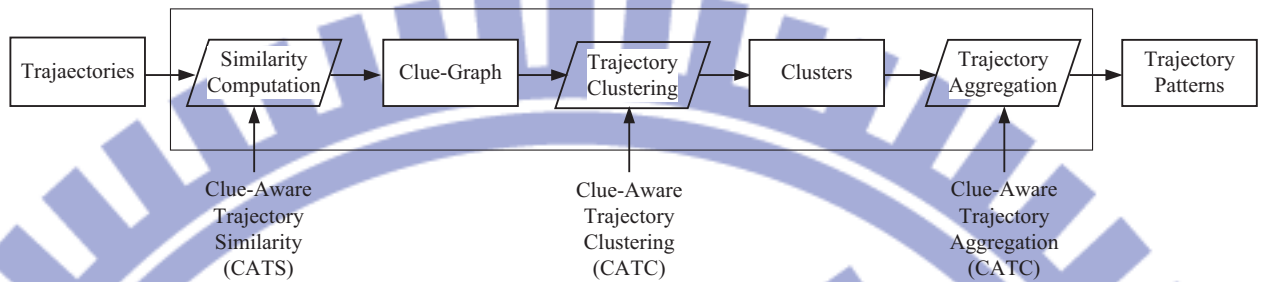


Figure 4.6: Overview of our proposed framework CACT.

4.4.1 Characteristics of Trajectories

For trajectory clustering, a similarity measurement between two trajectories should be formulated. Some characteristics of trajectories are listed as follows:

- Spatial and temporal bias:** In practice, trajectories are obtained by positioning devices. Generally speaking, data points of trajectories have both the spatial and temporal bias. For example, the position accuracy of GPS (Global-Position System) has spatial bias. For the temporal bias, the occurrence time of data points that capture exactly the same movement behavior is not always the same. Consider a worker goes to his office from his home at 8:00am every day. Data points of trajectories that record his movement behavior will not have the same occurrence time. One reason is that the positioning device needs some time to determine his location. Thus, even if this user leaves his home at 8:00am every day, data points of trajectories have some temporal bias.
- Local temporal shifting:** Due to movement speeds of users or time delay of user movements, trajectories may have local temporal shifting. For example, although a user follows the same movement path to his office every day, his daily trajectories may have sub-trajectories whose occurrence time

is shifted.

- **Noise:** Positioning devices can be easily affected by environment factors, such as buildings, shelters, and weathers. Hence, data points of trajectories usually have some *noises*.
- **Silent duration:** The length of a trajectory is mainly decided by the time the sampling rate. For the same movement path, even we set the same sampling rate for the positioning device, trajectories collected may have different lengths. This results from environmental factors (e.g., the weather), and the constraint of position devices (i.e., the capability of positioning devices in computing and networking). In this paper, a silent duration refers a time duration when there are no any data points available about user movements. For example, if we set the data points are collected every 5 seconds, at some time slots, some data points are lost due to the constraint of positioning devices or other environmental factors. On the other hand, to save energy of positioning devices powered by batteries, we may set a lower sampling rate. Consequently, a trajectory has a rough user movement behavior.

4.4.2 Design of Clue-Aware Trajectory Similarity

Given two trajectories, the design of the Clue-Aware Trajectory Similarity (abbreviated as CATS) is to capture as many similar data points between two trajectories as possible while still guaranteeing the given thresholds in the spatial and temporal domains. CATS is able to overcome the effects raised by characteristics of trajectories since CATA has a spatial decaying function, clue scores of data points and a new mapping scheme. The properties of CATS will be discussed later.

Since trajectories usually contain spatial bias and shifting, we first use a spatial decaying function to measure the degree of bias and shifting as follows:

Definition. Spatial Decaying Function: Given a spatial threshold ϵ , and two data points $p_{i,\ell} = (l_{i,\ell}, t_{i,\ell})$ and $p_{j,k} = (l_{j,k}, t_{j,k})$ from two trajectories (i.e., T_i and T_j), a spatial decaying function for two points $p_{i,\ell}$ and $p_{j,k}$ is defined as

$$f_{\epsilon}(p_{i,\ell}, p_{j,k}) = \begin{cases} 0, & \text{if } dist(p_{i,\ell}, p_{j,k}) > \epsilon \\ 1 - \frac{dist(p_{i,\ell}, p_{j,k})}{\epsilon}, & \text{otherwise} \end{cases}$$

where $dist(\cdot)$ is Euclidean distance between two data points.

The value of spatial decaying function is ranged from 0 to 1. Obviously, the closer the two data points, the larger the value is. Once the locations of two data points are exactly the same, the value is 1. On the other hand, once the distance between two points is far from ϵ , the value is 0. For example, Figure 4.7 shows two trajectories T_1 and T_2 , where the underlying grey lines are actual movements and the circles represent the data points of T_1 and T_2 . Consider $\epsilon = 4$ and Euclidean distance as the distance function. Given two points $p_{1,2} = (2, 2, 3)$ and $p_{2,1} = (3, 3, 3)$, it can be derived that $f_4(p_{1,2}, p_{2,1}) = 1 - \frac{\sqrt{2}}{4} = 0.65$. On the other hand, given $p_{1,4} = (7, 4, 9)$ and $p_{2,1} = (3, 3, 3)$, we can derive that $f_4(p_{1,4}, p_{2,1}) = 0$ since $dist(p_{1,4}, p_{2,1}) > \epsilon = 4$.

In the spatial decaying function, a parameter ϵ is given to tolerate the spatial bias and shifting of data points³. According to the Euclidean distance between two points, the spatial decaying function performs a continuous space quantization (i.e., from 0 to 1). The continuous space quantization could reflect the close degree of two data points compared to the discrete space quantization adopted by LCSS and EDR. For example, if the ϵ is set as 10 meters, consider two cases that two points with their distance 1 meter and two points with their distance 9 meters. In the clue-similarity measurement, the former case contains a stronger clue than the

³Users could set the parameter ϵ according to application requirements or trajectory patterns mined.

latter case. LCSS and EDR could not distinguish these two cases since these two points in each case have their distance smaller than 10 meters.

According to the spatial and temporal information of data points, we need to give a score for data points with respect to a trajectory. Note that a clue score of a data point is used to evaluate the clue degree of data points by mapping this data point to the data points of trajectories. Given a data point $p_{i,\ell} \in T_i$ and a trajectory T_j , a clue score of data point $p_{i,\ell}$ with respect to trajectory T_j is to identify the best mapping point of T_j to $p_{i,\ell}$ that is close to the given data point in the spatial and temporal domains.

Definition. Clue Score of Data Points: Given a point $p_{i,\ell}$, a trajectory T_j , a spatial threshold ϵ , and a temporal threshold τ , the clue score of this data point $p_{i,\ell}$ to trajectory T_j is defined as $score_{\epsilon,\tau}(p_{i,\ell}, T_j) = \max\{f_\epsilon(p_{i,\ell}, p_{j,k}) | p_{j,k} \in T_j \text{ and } t_{j,k} \in [t_{i,\ell} - \tau, t_{i,\ell} + \tau]\}$.

Assume that $\epsilon = 4$, $\tau = 4$. An example of trajectories is shown in Figure 4.7(a), where the underlying grey line is the real movement. Consider a data point $p_{1,5}$ of T_1 as an example. The clue score of data point $p_{1,5}$ with respect to trajectory T_2 is to find the best mapping data points of T_2 within a time interval $14 - 4$ to $14 + 4$. It can be verified in Figure 4.7(a) that four data points $p_{2,3}$, $p_{2,4}$, $p_{2,5}$ and $p_{2,6}$ of T_2 are possible mapping data points for $p_{1,5}$ since their time are within $14 - 4$ to $14 + 4$. Since $f_4(p_{1,5}, p_{2,5}) = 1 - \frac{\sqrt{5}}{4} = 0.44$ is the largest value among that of other points, the clue score of $p_{1,5}$ with respect to trajectory T_2 is thus $score_{\epsilon=4,\tau=4}(p_{1,5}, T_2) = 0.44$.

From the definition of clue scores, a temporal parameter τ is used to retrieve data points of trajectories whose occurrence times are within a particular time interval. Using this parameter, our proposed CATS can deal with local temporal

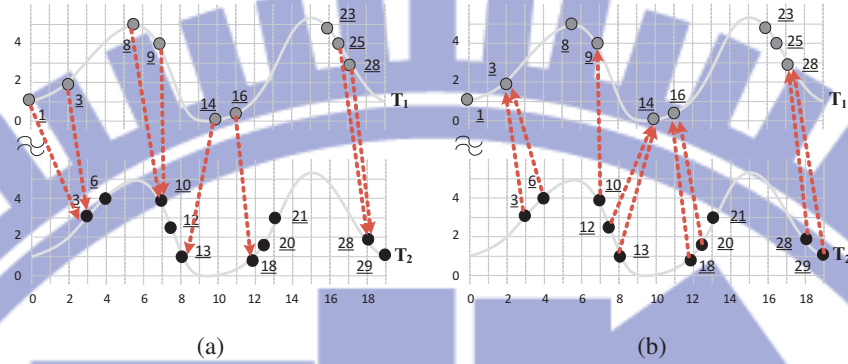


Figure 4.7: An illustrative example for clue-aware similarity of T_1 and T_2 .

shifting. Since the time of data points to be mapped are constrained by the temporal parameter, the clue score is sensitive to time. The clue score of data points with respect to a trajectory is ranged from 0 to 1. Clearly, the clue score of data points considers both the spatial and temporal information, while stilling allowing some tolerable spatial and temporal thresholds to overcome spatial and temporal bias.

In light of the clue score, we could define the clue similarity between two trajectories as follows:

Definition. Clue-Aware Trajectory Similarity: Given a spatial threshold ϵ and a temporal threshold τ , the clue-aware trajectory similarity from T_i to T_j is defined as $CATS_{\epsilon,\tau}(T_i, T_j) = \frac{1}{|T_i|} \times \sum_{p_{i,\ell} \in T_i} score_{\epsilon,\tau}(p_{i,\ell}, T_j)$.

For example, let $\epsilon = 4$ and $\tau = 4$. The arrows in Figure 4.8(a) show the mapping relationships from each data point of T_3 to data points of T_4 . Consequently, the clue-based similarity measurement from T_3 to T_4 is derived as $CATS_{4,4}(T_3, T_4) = \frac{1}{8} \times (score(p_{3,1}, T_4) + score(p_{3,2}, T_4) + score(p_{3,3}, T_4)) = \frac{1}{8} \times (1 + 1 + 0.25) = 0.28$.

4.4.3 Properties of Clue-based Similarity Measurements

From the definition of clue scores, both ϵ and τ thresholds are used to overcome spatial and temporal biases. Moreover, these two thresholds could deal with the local shifting scenario in both the spatial and temporal domains. Since noise data will have larger distance value, our spatial decay function can easily filter out noise data. For the mapping scheme, our clue-based similarity measurement allows many data points to map the same data point of other trajectories. Consider the clue-based similarity from T_i to T_j as an example. It is possible that some data points of trajectory T_i map to the same data point of T_j if the mapped data point of T_j fulfills both the spatial and temporal thresholds ϵ and τ . This mapping scheme is referred as n-to-1 mapping (abbreviated as n-1). Since data points of T_i have higher clue scores to T_j , these data points of T_i in fact provide a more detailed movement information for T_j . Thus, these data points are very helpful for the silent duration of T_j . Furthermore, due to the spatial and temporal thresholds, data points of T_i may not map any data points of T_j . In that case, the clue score is set to 0. Hence, CATS will reflect those sub-trajectories that have higher clue scores. Suppose that two trajectories have different lengths and these two trajectories have some clues. Our CATS will still derive a higher clue score for these two trajectories, showing that CATS is able to overcome silent durations of trajectories.

Note that CATS has an asymmetry property. For example, in Figure 4.8, $CATS_{4,4}(T_3, T_4) = 0.28$ is not equal to $CATS_{4,4}(T_4, T_3) = 0.76$. From the aforementioned example, it can be seen that T_3 provides only a limited number of clues for the movement behaviors. On the other hand, with a large CATS values, trajectory T_4 can provide more detailed movement information for T_3 . Hence, this asymmetry property of CATS is helpful for identifying relationships between two

trajectories. From our observations, there are two relations between two trajectories: the first one is that both of two trajectories can provide sufficient clues to each other. The second case is that from perspective of T_i , trajectory T_i is likely to provide some detailed sub-trajectories to trajectory T_j , but from the perspective of T_j , T_j does not have similar movement behavior with T_i . The asymmetric property reflects two kinds of relationships between two trajectories. The first relationship usually happens when silent durations of two trajectories are distributed in the similar way and two trajectories have only some spatial and temporal shifting. For the first case, these two trajectories can be recognized as one movement behavior. Figure 4.7 shows an illustrative example, where both T_1 and T_2 have the same amount of clues to each other. Thus, two trajectories are very likely to follow the same movement behavior. For the second relationships, one trajectory may contain the other trajectory. For example, in Figure 4.8, the actual movement of T_3 and T_4 are the same (i.e., the underlying grey lines). Figure 4.8(a) shows that mapping data points of T_4 have clues to T_3 . However, in Figure 4.8(b), most data points of T_3 have no clues to T_4 . In this case, we can see that by compensating T_3 with the data points of T_4 , the movement behavior can be revealed in more detail. Consequently, with the asymmetry property, we further propose a clue-aware clustering algorithm to cluster trajectories into several groups, where each group represents one frequent movement behavior.

4.5 Clue-Aware Trajectory Clustering Algorithm

In this section, we propose algorithm CATC (standing for Clue-Aware Trajectory Clustering), to cluster trajectories into several clusters and each cluster represents one frequent movement behavior.

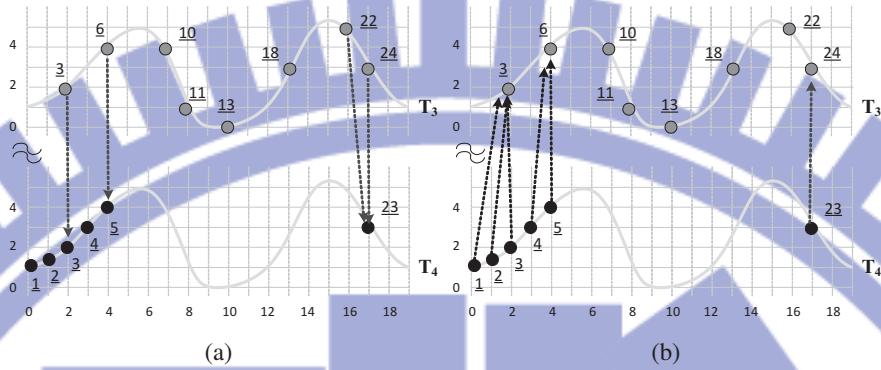


Figure 4.8: An example to show the asymmetric property of CATS.

4.5.1 Design of Clue-Aware Trajectory Clustering Algorithm

The overview of our proposed clue-aware trajectory clustering algorithm is outlined as follows:

Procedure of CATC

Step 1. Clue-Graphs generation phase: In this phase, a graph structure is used to represent clue-similarity measurements among trajectories.

Step 2. Core set identification phase: A core set refers to a set of trajectories that have strong clue similarities with each other. In this phase, given a clue-graph, we discover a set of core sets.

Step 3. Cluster discovery phase: In this phase, we merge core sets as a cluster and extract clusters that have a sufficient number of trajectories as frequent movement behaviors.

Given a set of trajectories, we first determine the values of clue-aware trajectory similarities among them. Then, we utilize a graph structure to represent their clue-aware trajectory similarities. The definition of a clue-graph is given below:

Definition. Clue-Graph: Given a set of trajectories $T = \{T_1, T_2, \dots, T_n\}$ and a threshold λ , a clue-graph is a weighted directed graph $G = (V, E)$. In the clue-

graph G , a set of vertices $V = \{v_1, v_2, \dots, v_n\}$ represents the set of all trajectories and a set of edges is defined as $E = \{(v_i, v_j) | CATS_{\epsilon, \tau}(v_i, v_j) \geq \lambda\}$ with their weight as $CATS_{\epsilon, \tau}(v_i, v_j)$.

Figure 4.9 shows a clue-graph with $\lambda = 0.4$. A clue-graph is used to represent the clue relations between trajectories. There will be no edge between two vertices if the corresponding trajectories do not have significant clues (i.e., their clue similarity value is smaller than λ). In the clue-graph, we could further define a directly clue-reachable relationship among trajectories as follows:

Definition. Directly clue-reachable: A vertex v_i is directly clue-reachable to a vertex v_j , denoted as $v_i \rightsquigarrow v_j$, if $(v_i, v_j) \in E$.

As pointed out early, our proposed similarity CATS has an asymmetry property. It is possible that one trajectory has significant clues to the other one but the reverse does not hold. On the other hand, if two trajectories could provide clues to each other, these two trajectories likely infer the same movement behavior. In this case, these two trajectories are directly clue-reachable to each other in the clue-graph, i.e., they are one-hop neighbors. To represent the above scenario, we define a core set as follows:

Definition. Core set: Given a clue-graph $G = (V, E)$, a core set is a directed complete subgraph of G , where any two nodes v_i and v_j in a core set are directly clue-reachable to each other.

Each node in the core set has highly clues indicating that these nodes in this core set capture the same movement behavior. Clearly, these core sets are viewed as seeds and these seeds could further expand nearby seeds in the clue-graph for possible merging. The detailed merging procedure among core sets is pre-

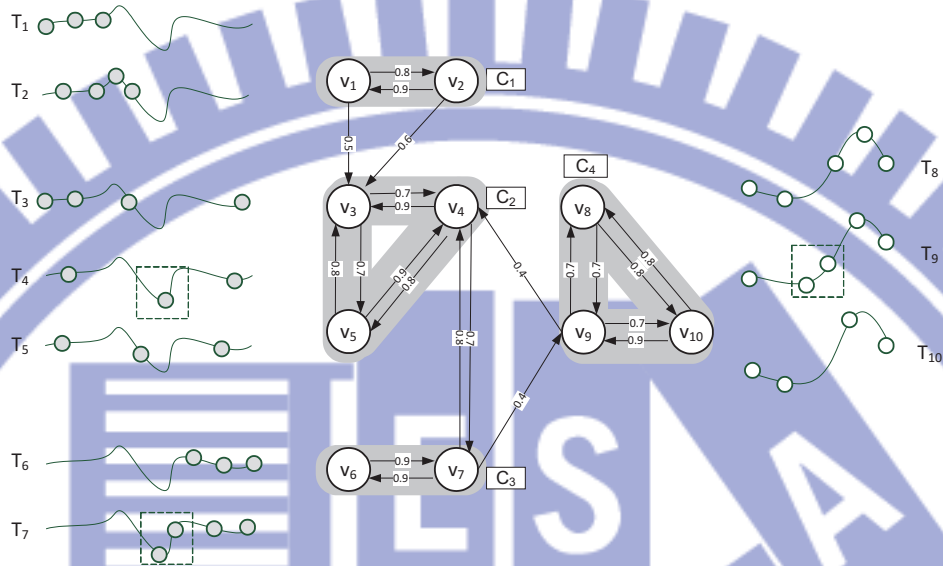


Figure 4.9: An illustrative example for CATC.

sented later. To identify core sets in the clue graph, we borrow existing works of clique-covering algorithms. Most clique-covering algorithms are executed in an undirected graph [20]. To facilitate the generation of core sets by using existing clique-covering algorithms, a strong clue-graph, denoted as SC-graph, from the clue-graph G is defined as follows:

Definition. Strong Clue-graph: Given a clue-graph $G = (V, E)$, a Strong Clue-Graph, denoted as SC-G, is a undirected graph and represents as $SC-G = (V, E')$, where $(v_i, v_j) \in E'$ if both $(v_i, v_j) \in E$ and $(v_j, v_i) \in E$.

By performing an existing clique-covering algorithm in SC-G, a set of core sets is derived. Then, nodes in the same core set are labeled in the clue-graph as well. A minimum clique cover for the clue-graph in Figure 4.9 are $\{C_1, C_2, C_3, C_4\}$, where vertices in the same shade region belong to a clique.

After the generation of core sets, these core sets may merge other core sets if two core sets have some reachable relationships. Since a core set may have reachable relationships with more than one core set, one should judiciously decide which core sets are selected for merging. To infer whether two core sets capture the same movement behavior or not, the number and the weights of edges between two core sets should be considered. Intuitively, if both the number of edges among vertexes of two core sets and edge weights are larger, these two core sets are likely to reflect the same movement behavior. Furthermore, two core sets may still have clue reachable relationships via other core sets between these two core sets. To define the clue reachable relationship among two core sets (referring to as clue-connected), we should define a clue-reachable relationship between two vertex as follows:

Definition. Clue-reachable: A vertex u is clue-reachable to a vertex v , denoting as $u \rightsquigarrow^* v$, if there exists a chain of vertices $v = v_1, v_2, \dots, v_n = u$ such that $v_i \rightsquigarrow v_{i+1}$ for all $i = 1, 2, \dots, n - 1$.

With the definition of clue-reachable, we could define the clue-connected relationship between two core sets as follows:

Definition. Clue-connected: Given two core sets C_u and C_v , C_u can clue-connect to C_v , denoted as $C_u \Rightarrow C_v$, if there exists a core set C_w such that $x \rightsquigarrow^* y$ for all $x \in C_u$ and for some $y \in C_w$, and $y' \rightsquigarrow^* z$ for all $y' \in C_w$ and for some $z \in C_v$.

For example, in Figure 4.9, C_1 can clue-connect to C_2 . The reason is that given $C_v = C_1$, $C_w = C_1$, and $C_u = C_2$, we have all vertices in C_1 are clue-reachable to some vertices in C_1 since C_1 is a core set, and $v_1 \rightsquigarrow^* v_3$ and $v_2 \rightsquigarrow^* v_3$. Two clue-connected core sets demonstrate the same movement behavior if these two core sets have connected core sets between these two core sets. Clearly, if two

core sets have a clue-connected relationship, these two core sets should be put in the same cluster.

Considering the aforementioned definitions, the cluster results of algorithm CATC should be a set of clusters consisting of core sets and the number of vertices in each cluster should be larger than min_sup . To facilitate our presentation, a candidate cluster is used to represent our merging results of core sets. Initially, each core set is viewed as one candidate cluster. We will iteratively merge candidate clusters until no further merge operation is needed. One criterion stopping this merge operation is to measure the quality of cluster results. To evaluate the quality of cluster results, we have the following definitions: clue-cohesion and clue-separation.

Definition. Clue-Cohesion: Given a candidate cluster K , the clue-cohesion of K , denoted $CCOH(K)$, is defined as the minimum weight that for every core set $C_i \in K$, there exists a core set $C_j \in K$ such that $C_i \Rightarrow C_j$.

Definition. Clue-Separation: Given two candidate clusters K_m and K_n , the clue-separation from K_m to K_n , denoted $CSEP(K_m, K_n)$, is defined as the total weights of all edges from K_m to K_n .

For example, consider C_2 and C_4 in Figure 4.9. If we want to make $C_4 \Rightarrow C_2$, two extra edges (e.g., (v_8, v_3) and (v_{10}, v_5)) with total weight $2 \times \lambda = 0.8$ should be added since there already exists an edge (v_9, v_4) from C_4 to C_2 . Thus, if a cluster K contains C_2 and C_4 , the clue-cohesion of this cluster can be derived as $CCOH(K) = 0.8$. The clue-separation $CSEP(C_4, C_2) = 0.4$ since there is one edge (v_9, v_4) from C_4 to C_2 with total weight $\lambda = 0.4$. Note that the clue-cohesion $CCOH(K_m, K_n)$ is zero if K_m can clue-connect to K_n . As such,

two clue-connected clusters are easily merged, which fits our goal that two clue-connected core sets should be put into the same cluster.

According to the quality measurements of clusters, we intend to have a set of clusters such that the cluster result should have smaller clue-cohesions and clue-separations among clusters. In the other words, trajectories within the same cluster have as many clues as possible and trajectories between different clusters have as few clues as possible. Therefore, the desired cluster results could be defined as follows:

Definition. Clusters: Given a clue-graph $G = (V, E)$, derive a set of clusters $\mathcal{K} = \{K_1, K_2, \dots, K_m\}$ such that (1) K_i contains a set of core sets, (2) minimize $\sum_{K_m \in \mathcal{K}} CCOH(K_m) + \sum_{K_m, K_n \in \mathcal{K}} CSEP(K_m, K_n)$, and (3) $|K_i| \geq min_sup$ for all $K_i \in \mathcal{K}$.

In light of the requirements of clusters, we design a benefit function that takes both the clue-cohesion and clue-separation into accounts in merging candidate clusters. The benefit function is formulated as follows:

Definition. Benefit Function: Given two candidate clusters K_m and K_n , the benefit function is defined as $Benefit(K_m, K_n) = DesCSEP(K_m, K_n) - IncCCOH(K_m, K_n)$. $DesCSEP(K_m, K_n) = (CSEP(K_m, K_n) + CSEP(K_n, K_m))/2$ and $IncCCOH(K_m, K_n) = CCOH(K_m) + CCOH(K_n) + \sum_{C_i \in K_m} \min_{C_j \in K_n} \{I(C_i, C_j) \times \lambda\}$, where $I(C_i, C_j)$ denotes the number of vertices that have no edge from core set C_i to C_j .

Generally speaking, merging two candidate clusters will increase the total clue-cohesion while decreasing the total clue-separation. Therefore, merging two

clusters can minimize the sum of cohesion and separation if this merging could lead to a larger amount of decreased separation than a smaller amount of increased cohesion. Consequently, the benefit function is to evaluate whether merging two candidate clusters is able to reduce the value of the objective function (i.e., the sum of the total clue-cohesion and the total clue-separation). Assume that we intend to merge two candidate cluster K_m and K_n . The first term of the benefit function (i.e., DesCSEP) represents how many clue-separation could be reduced by merging K_m and K_n . The average of the clue-cohesions between two candidate clusters aims to prevent the scenario that only one cluster has a lot of edges to the other one but there's no edges in reverse. The second term of the benefit function (i.e., IncCCOH) is to evaluate the amount of increase in clue-cohesion by merging K_m and K_n . For two candidate clusters K_m and K_n , they need $CCOH(K_m)$ and $CCOH(K_n)$ to make their core sets clue-connected. The last term of IncCCOH refers to the minimum weight that every core sets in K_m can clue-connect to some core set in K_n . Exploiting this benefit function, algorithm CATC iteratively selects two candidate clusters with the maximum benefit value until the value of the benefit function is smaller than zero. Once the merging operation is finished, candidate clusters that have more than min_sup vertices will become final clusters. The reason for having at least min_sup vertices is that each cluster represents one frequent movement behavior. Note that the computation of the benefit function could be implemented by dynamic programming strategy efficiently: since candidate clusters are expanded in a bottom-up fashion, the clue-cohesions and clue-separations of any two candidate clusters are computed at previous rounds. Therefore, we could explore a dynamic programming strategy in algorithm CATC.

4.5.2 Running Example for Clustering Discovery in algorithm CATC

The execution scenario of CATC could be best understood in Figure 4.9. In this figure, T_1 to T_7 capture the first movement behavior and T_8 to T_{10} record the second movement behavior. Note that there are edges from v_4 to v_9 , and from v_7 to v_9 . The reason is that they have some nearby points as shown in the dashed square such that there are some clues between them. In the beginning, CATC finds four core sets $C = \{C_1, C_2, C_3, C_4\}$ and the vertices in the same shaded area are in the same clique. Each clique is a candidate cluster in \mathcal{K} , i.e., $\mathcal{K} = \{K_1 = C_1, K_2 = C_2, \dots, K_4 = C_4\}$. To evaluate whether two candidate clusters can be merged, we first compute benefit values for each pair of candidate clusters. For example, it could be computed that $Benefit(K_1, K_2) = (0.5 + 0.6)/2 + 0 = 0.55$ (since all vertices in K_1 are clue-reachable to v_3), $Benefit(K_3, K_2) = (0.7 + 0.8)/2 - 0.4 = 0.35$ (since v_6 in K_3 is not clue-reachable to any vertex in K_2), and $Benefit(K_4, K_2) = (0.4 + 0.4)/2 - 2 \times 0.4 = -0.4$ (since v_8 and v_{10} in K_4 are not clue-reachable to any vertex in K_2). Since the $Benefit(K_1, K_2)$ is the maximum, two candidate clusters K_1 and K_2 are merged into one larger candidate cluster $K'_1 = \{K_1, K_2\}$. Next, CATC continues to compute the benefit values between all pairs of candidate clusters. Specifically, if we intend to merge K_3 to K'_1 , we could first derive $DesCSEP(K_3, K'_1) = (0.7 + 0.8)/2 = 0.75$. Then, $C_3 \in K_3$ needs the cost $1 \times 0.4 = 0.4$ to clue-connect to $C_2 \in K'_1$. Thus, $IncCCOH(K_3, K'_1) = CCOH(K_3) + CCOH(K'_1) + 0.4 = 0 + 0 + 0.4 = 0.4$. Since $Benefit(K_3, K'_1)$ is the maximum, CATC merges K_3 and K'_1 into a new candidate cluster $K'_1 = \{C_1, C_2, C_3\}$. Finally, since all the benefit values are negative, CATC terminates. Assume that the threshold min_sup is 4, then K_4 is deleted since the number of trajectories in K_4 is smaller than 4. Finally, one cluster $K'_1 = \{C_1, C_2, C_3\}$ is found.

Time Complexity: In line 1, CATC constructs a clue-graph which requires $O(N^2)$. In line 2 to line 3, a clique covering algorithm is performed in a clue-graph, which takes $O(N^2)$ [20]. In line 5 to line 8, initializing table CCOH, CSEP, and I costs $O(N^2)$. The outer loop from line 9 to line 27 depends on the benefit values. The worst case is that every single vertex is a clique such that this loop executes at most $N - 1$ times. From line 11 to line 24, there are a nested loop. From line 14 to line 18, all pairs of core sets are enumerated such that there are totally $C(\ell, 2)$ combinations. From line 24 to line 26, updating CCOH table needs $O(1)$ -time, and updating the CSEP table requires $O(\ell)$ -time. Therefore, the outer loop from line 7 to line 27 totally takes at most $C(N - 1, 2) + C(N - 2, 2) + \dots + C(2, 2) = O(N^3)$. To sum up, time complexity of CATC is at most $O(N^3)$.

4.6 Clue-Aware Trajectory Aggregation Algorithm

Given a set of clusters, for each cluster, we intend to derive hot regions and a sequence of hot regions is thus represented as one frequent movement behavior. As pointed out early, each hot region has its representative line segment. In this section, we propose a Clue-Aware Trajectory Aggregation algorithm (abbreviated as CATA) to derive a set of representative line segments. In algorithm CATA, we first determine a candidate line segment within one core set and then candidate line segments from different core sets will be aggregated as representative line segments.

4.6.1 Determine Candidate Line Segments within a Core Set

In a core set, a trajectory with the maximal sum of edge weights is selected as a base trajectory. Intuitively, the base trajectory selected contains more clues to other trajectories within a core set. Based on the base trajectory, data points from other trajectories within the same core set are selected for deriving candidate line

Algorithm 6: CATC: Clue-Aware Trajectory Clustering Algorithm

Input : Trajectories: $\mathcal{T} = \{T_1, T_2, \dots, T_N\}$; Thresholds: λ, min_sup
Output : Set of clusters: \mathcal{K}

- 1 Construct a clue-graph $G = (V, E)$ by \mathcal{T} and λ ; // Step 1
- 2 Construct a Strong clue-graph $SC-G = (V, E')$ from G ; // Step 2
- 3 $C = \{C_1, C_2, \dots, C_\ell\} \leftarrow$ a clique cover of $SC-G$;
- 4 $\mathcal{K} \leftarrow C$;
- 5 **foreach** $(K_m, K_n) \in \mathcal{K} \times \mathcal{K}$ **do**
- 6 $CCOH[K_m] \leftarrow 0$;
- 7 $CSEP[K_m, K_n] \leftarrow$ total weights from K_m to K_n ;
- 8 Compute $I(K_m, K_n)$;
- // Step 3
- 9 **repeat**
- 10 $Benefit \leftarrow \infty$;
- 11 **foreach** $(K_m, K_n) \in \mathcal{K} \times \mathcal{K}$ **do**
- 12 $r \leftarrow -1$;
- 13 $Des \leftarrow (CSEP[K_m, K_n] + CSEP[K_n, K_m])/2$;
- 14 $Inc \leftarrow 0$;
- 15 **foreach** $C_i \in K_m$ **do**
- 16 $q \leftarrow \infty$;
- 17 **foreach** $C_j \in K_n$ **do**
- 18 $q \leftarrow \min(q, I(C_i, C_j))$;
- 19 $Inc \leftarrow Inc + q$;
- 20 $Inc \leftarrow CCOH[K_m] + CCOH[K_n] + q$;
- 21 $Benefit \leftarrow Des - Inc$;
- 22 **if** $Benefit > r$ **then**
- 23 $(S, T) \leftarrow (K_m, K_n)$;
- 24 $Benefit \leftarrow r$;
- 25 **if** $Benefit > 0$ **then**
- 26 $\mathcal{K} \leftarrow \mathcal{K} \cup \{S \cup T\} - S - T$;
- 27 Update $CCOH$ and $CSEP$;
- 28 **until** $Benefit < 0$;
- 29 Keep $K_m \in \mathcal{K}$ if K_m contains more than min_sup vertices;

segments. If data points from other trajectories are far away from the base trajectory (i.e., larger than ϵ), these data points are viewed as noise data points. Assume that two consecutive data point q and r are from the base trajectory and the occurrence time of r is larger than that of q . Suppose that a data point p is close to a line \overline{qr} , a directed line by connecting two consecutive data points q and r of a base trajectory, this data point p is useful to derive candidate line segments. In other words, if $\text{dist}(p, \overline{qr}) < \epsilon$, a data point p is fulfilled the spatial constraint and its occurrence time will be further investigated. If the occurrence time of p is between $[t_q - \tau, t_r + \tau]$, where the occurrence time of q and r is denoted as t_q and t_r , the occurrence time of data point p is revised as $t_p = t_q + (t_r - t_q) \times \frac{\text{dist}(q,p)}{\text{dist}(q,p) + \text{dist}(p,r)}$. On the contrary, if the occurrence time of data point p is not within $[t_q - \tau, t_r + \tau]$, data points p will not be considered. Consider an example in Figure 4.10(a), where the dotted line is a base trajectory and data points of other trajectories are marked as grey points associated with their occurrence time. As can be seen in Figure 4.10(a), grey point y is eliminated because $\text{dist}(y, \overline{cd}) > \epsilon$. For grey points w , x , and z , these data points are close to the base trajectory. We should further investigate their occurrence time. Suppose that $\tau = 2$ and the distance between a and w equals to that between b and w . Since the time of w is 7 which is between $[2 - 2, 6 + 2]$, the time of w is revised as $2 + (6 - 2) \times \frac{\text{dist}(a,w)}{\text{dist}(a,w) + \text{dist}(w,b)} = 4$. On the other hand, the time of x is 11 which is not within the interval $[6 - 2, 8 + 2]$. Thus, data point x is discarded. Given two trajectories in Figure 4.10(a), the set of data points is shown in Figure 4.10(b).

Once we have obtained a set of data points from trajectories within a core set, following the approach adopted in [5], we utilize the Douglas-Peucker algorithm [25] to determine candidate line segments. Douglas-Peucker algorithm is used to determine representative lines that capture the distribution of data points given. To focus our main theme of this paper, the detailed description for Douglas-

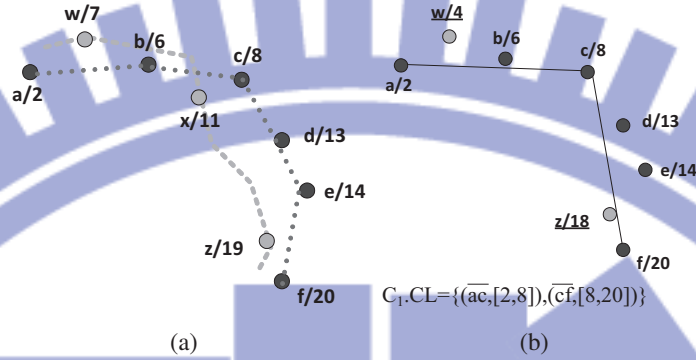


Figure 4.10: An example of generating candidate line segments in a core set.

Peucker algorithm can be viewed in [25]. Note that Douglas-Peucker just derives simplified lines from a set of spatial points. To consider the temporal information of data points, the spatiotemporal data reduction mechanism that is modified from Douglas-Peucker could be applied [6]. To facilitate the presentation of our paper, a core set C_i has its set of candidate line segments, denoted as $C_i.CL = \{(l_{i,1}, TI_{i,1}), (l_{i,2}, TI_{i,2}), \dots, (l_{i,k}, TI_{i,k})\}$, where $l_{i,k}$ is the k th line segment with its time interval $TI_{i,k}$. Given the data points in Figure 4.10(b), the candidate line segments determined are shown in Figure 4.10(b).

4.6.2 Generate Representative Line Segments within a Cluster

Since a cluster may have more than one core set, in this section, given candidate line segments derived by each core set within a cluster, a set of representative line segments with their time intervals will be generated. In a cluster, a core set C_i that connects to as many core sets as possible is selected. Then, the set of candidate line segment of the selected core set are viewed as the default set of representative line segments. By examining candidate line segments of other core sets within a cluster, we could further tune the set of representative line segments. The examination order is decided by the benefit function. As suggested, the benefit function

will be put together and then derive new representative line segments.

The procedure can be best understood by the example in Figure 4.11(a), where there are two core sets C_1 and C_2 with their sets of candidate line segments $C_1.CL$ and $C_2.CL$. Assume that C_1 has a set of candidate line segments from Figure 4.10(b) and the set of candidate line segments of C_1 is set to the default set of representative line segments. Figure 4.11(a) shows candidate line segments from core set C_2 with their time interval as $TI_{2,1} = [3, 10]$ and $TI_{2,2} = [15, 19]$. Then, given a time interval $TI_{2,1} = [3, 10]$, we could one representative line segment whose time interval (i.e., $TI_{1,1} = [2, 8]$) has some overlapped with $TI_{2,1} = [3, 10]$. Thus, data points along with candidate line segments are checked whether these data points are close to the representative line segment or not. In our example, point m is far from the representative line segment and thus is excluded. For the time interval $TI_{2,2} = [15, 19]$, a representative line segment with time interval $TI_{1,2} = [13, 20]$ is also retrieved. Similarly, data points of candidate line segments are verified whether they are far from the representative line segment or not. Finally, data points from candidate line segments and representative line segments are used to derive a new set of representative lines with their time intervals. In this example, we have a set of representative line segments as $\overline{c\bar{y}}$, $\overline{y\bar{d}}$ and $\overline{d\bar{f}}$ with their time intervals $[10, 12]$, $[12, 13]$, and $[14, 20]$, respectively. Once the set of representative line segments is determined, hot regions are easily decided by putting these representative line segments as central lines with bias range given. The time interval of each hot region is set to the time interval of the corresponding representative line segment.

Time Complexity: To facilitate analysis, the length of trajectories are supposed to be $|T|$. The loop from line 3 to line 10 executes $|K_i|$ times. In this loop, line 4 needs a linear scan to find T_r , which costs $O(|X|)$ time. The inner loop from line 5 to line 7 needs a linear scan for each trajectory for adjusting location and

time stamp, which costs $O(|X||T|)$ -time. Therefore, from line 3 to line 10, it costs $O(|K_i||T|)$. Note that the size of Ω is $|K_i|$ since line 9 adds a trajectory for each round. Therefore, from line 11 to line 15, it needs $O(|K_i| \times |T|\log|T|)$ -time since the Douglas-Peucker algorithm needs $O(|T|\log|T|)$. Therefore, the total time complexity should be $\sum_i O(|K_i||T| + |K_i||T|\log|T|) = O(|K||T|\log|T|)$.

Algorithm 7: Clue-Aware Trajectory Aggregation algorithm

Input : A set of clusters: \mathcal{K}
Output : Trajectory pattern: R

```

1 for each cluster  $K_i \in \mathcal{K}$  do
2    $\Omega \leftarrow \phi$ ;
3   for each core set  $X \in K_i$  do
4      $T_r \leftarrow$  the trajectory similar to most trajectories in  $X$ ;
5     for each other trajectory  $T$  in  $X$  do
6       Eliminate points of  $T$  with intolerable location and time stamp to  $T_r$ ;
7       Add remaining points of  $T$  into  $T_r$ ;
8     end
9     Add  $T_r$  into  $\Omega$ ;
10  end
11  while  $|\Omega| > 1$  do
12     $A \leftarrow$  representative line segments which core set is most clue-reachable from others;
13     $B \leftarrow$  representative line segments which have maximal values in benefit function;
14     $L \leftarrow$  Aggregate  $A$  and  $B$ ;
15  end
16  Make hot regions for trajectory patterns;
17 end

```

4.7 Performance Evaluation

We conduct experiments to evaluate our proposed algorithms on both real and synthetic datasets. In Section 4.7.1, our experimental settings are presented. Performance comparisons of our clue-aware similarity measurement and clue-aware clustering algorithm with previous works are then described. Moreover, trajectory patterns mined by our CACT and other works are visualized. Finally, sensitivity analysis on CACT is investigated in Section 4.7.5.

4.7.1 Experimental Environment

To facilitate our validation on mined trajectory patterns, both real and synthetic datasets are used. In the real dataset (that is, the CarWeb dataset), trajectories are selected from the *CarWeb* platform [43]. In the CarWeb dataset, we are aware of ground truth about movement behaviors such that verifying the quality of mined trajectory pattern is simple. We select trajectories that capture four types of frequent movement behaviors shown in Figure 4.12. Note that both Type 2 and Type 4 have similar movement paths but their time are different. Trajectories of Type 2 present a movement behavior happening on an early morning, whereas trajectories of Type 4 depict another movement behavior in the late afternoon. Table 4.2 shows statistic information about our selected trajectories, including the number of trajectories, the length of trajectories, the time duration of trajectories. Although some previously proposed datasets are available, such as hurricane and animal datasets used in [38], evaluating the quality of mining results is difficult because we do not have any domain knowledge on these datasets. For example, the Australian Sign Language dataset [8][57] which is related to hand-writing trajectory dataset is not generated by the positioning devices (for instance, GPS), and is not suitable for our experiments. Consequently, CarWeb dataset is to validate our mining results.

To examine the impact of trajectory characteristics on our proposed algorithms, we further generate a synthetic dataset, where trajectories are generated from a given set of seed trajectories. The number of data points in these source trajectories (referred to as the length of trajectories) is from 3000 to 6000, which is much longer than the length of real trajectories in CarWeb dataset. For each seed trajectory, we simulate synthetic trajectories with spatial and temporal bias. Specifically, for data point p from a seed trajectory, a data point will have a prob-



Figure 4.12: Trajectories in *CarWeb* dataset.

ability of being a bias data point. The probability value is set to P_{bias} . If the data point generated is a bias data point, the location of this bias data point will far away the location of data point p at most r meters and its occurrence time is shifted at most t minutes from the occurrence time of p . Thus, we could derive a synthetic trajectory with spatial and temporal bias from a seed trajectory. To investigate the effect of silent durations, two parameters are used to control the distribution of silent durations. Specifically, given a seed trajectory, a synthetic trajectory is generated every SI seconds, where SI is the sampling interval. For example, in the *CarWeb* dataset, all trajectories are generated every 5 seconds. With a large sampling interval, the connective data points may far away, leading a larger silent duration. Another parameter P_{clue} is the probability of reporting location points. Note that although data points of a trajectory will be generated every SI seconds, each data point reports its location data with P_{clue} probability. Consequently, if SI is smaller, a smaller P_{clue} still leads to more silent durations in a trajectory. The default values of parameters are $\lambda = 0.3$, $\tau = 300$ seconds, $\epsilon = 100$ meters, $min_sup = 0.1$, $r = 300$ meters, $t = 300$ seconds. The value of ϵ is decided by averaging the length of silent durations of all trajectories.

	Type 1	Type 2	Type 3	Type 4
Number of trajectories	14	12	5	12
Average length	634	128	341	142
Standard deviation in lengths	130.14	28.39	185.95	30.21
Average time duration	62	12	49	14
Standard deviation in time durations	9.27	1.98	42.26	2.1

Table 4.2: Statistic information about the selected trajectories in CarWeb dataset.

4.7.2 Performance Comparison of Similarity Measurements

In this section, we examine the performance comparison among different similarity measurements. Our competitors are LCSS, DTW, wDF, and EDR. In all approaches, if setting spatial and temporal thresholds are needed, the spatial threshold is set as 100 meters and temporal threshold is set as 300 seconds. To compare the effectiveness of different distance functions, we borrow the evaluation method in [14] to utilize the one nearest neighbor (1NN) classifier. In the CarWeb dataset, each trajectory is labeled by its movement behavior (i.e., one of four movement types). Given a specific similarity measurement, the 1NN classifier is to predict the label of a trajectory as its nearest neighbor measured by the given similarity measurement from the training datasets. Since CATS is asymmetric relationship, the nearest neighbor in CATS is designed. Assume that we want to find the nearest neighbor of a trajectory T_i . All other trajectories would be mapped in a 2D plane. Explicitly, a trajectory T_j could be represented as a point $(CATS(T_i, T_j), CATS(T_j, T_i))$ in this 2D plane. Then, the nearest neighbor of a trajectory T_i in CATS is the skyline point T_j with the maximum value of $(CATS(v_i, v_j) + CATS(v_j, v_i))^4$.

The classification error rate is defined as the ratio between the number of incorrect prediction and the total number of trajectories. Because the performance

⁴The reason is that among the skyline points, we choose the trajectories that have the strongest clues with T_i as the closest neighbor of T_i .

of the 1NN classifier is highly sensitive to the given similarity measurement, the classification error rate could directly reflect the effectiveness of similarity measurements. Based on the 1NN classifier, we adopt the cross-validation approach in [14] in which training data is randomly divided into k subsets. Then, one subset is selected for testing and the other of $k - 1$ subsets are used as training sets. Finally, the average error rate of the 1NN classifier over the k cross validation is reported.

Figure 4.13 shows experimental results of different similarity measurements. Figure 4.13(a) depicts the experimental results with three types of trajectories (i.e., Type 1, Type 2 and Type 3). This figure shows that CATS and EDR have lower average error rates. Specifically, wDF and LCSS fail to predict the correct labels because most trajectories pass many nearby locations. DTW does not perform effectively due to the large standard deviations of Type 1 and Type 3 trajectories. Because EDR accounts for not only common sub-trajectories but also gaps between trajectories, EDR can clearly distinguish trajectories with different types. Moreover, the average error rates of EDR are slightly smaller than CATS. If Type 4 trajectories are included, CATS can outperform EDR. Note that Type 2 and Type 4 trajectories have similar movements with different occurring time. Due to time sensitivity, CATS can easily distinguish Type 2 and Type 4 trajectories compared with EDR. This experiment also shows the advantage of CATA in which movement behaviors with different time are distinguished.

Next, the effects of silent durations are investigated. First, we study the scenario involving the silent durations being evenly distributed in trajectories. Figure 4.14(a) shows the results when the SI is set to 10, 20, and 30 seconds. In all cases, CATS and LCSS outperform other approaches. Except for wDF and DTW, the average error rates of the other approaches remain almost constant when SI increases. Because the mapping scheme of DTW and wDF requires locating the nearest mapping points, silent durations may significantly affect the values of

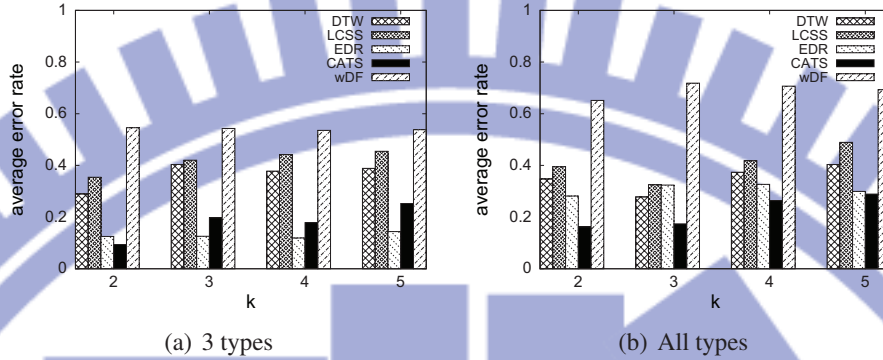


Figure 4.13: Average error rates with k varied.

DTW and wDF. Now, we investigate the impact of P_{clue} to the randomness of silent durations. Figure 4.14(b) shows that when P_{clue} is 70%, the average error rates of DTW and wDF increase slightly and those of CATS, LCSS and EDR decrease slightly. However, when P_{clue} decreases to 50%, the average error rates of all similarity measurements significantly decrease. The reason could be that there are many nearby locations of all types of trajectories, especially those places around the starting points. These locations may let all similarity measurements to make inaccurate predictions. When P_{clue} decreases, these nearby locations becomes silent durations more easily. Consequently, the number of these locations is reduced. Therefore, this phenomenon benefits similarity measurements in distinguishing different types of trajectories.

4.7.3 Performance Comparison of Clustering Algorithms

Performance comparisons of clustering algorithms are examined. We compare our proposed clustering algorithm CATC with traditional clustering algorithms (i.e., DBSCAN and Hierarchical clustering) and existing trajectory clustering algorithms. *Entropy* and *purity* are used to measure the quality of clustering results, where entropy is a function of the distribution of classes in the resulting clusters,

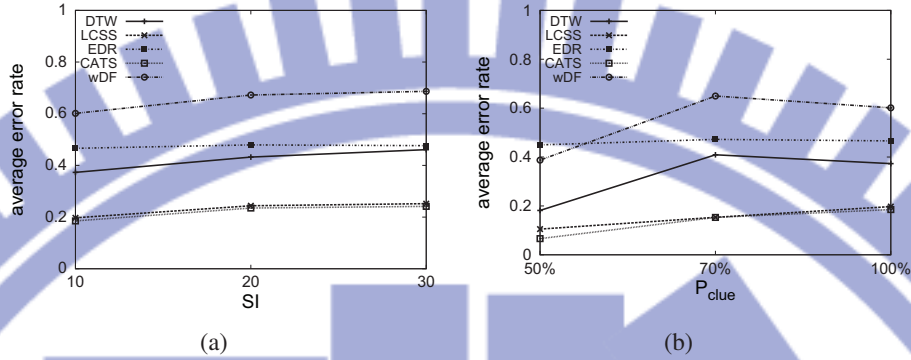


Figure 4.14: Average error rates with varying the distribution of silent periods.

and purity is a function of the relative size of the largest class in the resulting clusters. Given a particular cluster, S_r , of size n_r , the entropy of this cluster is defined as $E(S_r) = -\frac{1}{\log(q)} \sum_{i=1}^q \frac{n_r^i}{n_r} \log \frac{n_r^i}{n_r}$, where q is the number of classes in the dataset, and n_r^i is the number of trajectories of the i th class that are assigned to the r th cluster. The entropy of the entire clustering solution is then defined as $\sum_{r=1}^k \frac{n_r}{n} E(S_r)$. In general, the smaller the entropy value, the more favorable the clustering solution is. Similarly, the purity of a cluster is defined as $P(S_r) = \frac{1}{n_r} \max_i(n_r^i)$. Thus, the overall purity of the clustering solution is formulated as $\sum_{r=1}^k \frac{n_r}{n} P(S_r)$. The larger the value of purity, the more favorable the clustering solution is.

To compare our proposed algorithm with traditional clustering algorithm (i.e., DBSCAN and hierarchical clustering), the same similarity measurement is used for fair comparison. Thus, the similarity measurement for DBSCAN and hierarchical clustering is CATS. Note that the distance function used in DBSCAN and hierarchical clustering is required to be symmetric. Due to the asymmetry of CATS, the distance function for two trajectories (e.g., T_i and T_j) is slightly revised as the average sum of two directed edges: $d(T_i, T_j) = 1 - \frac{CATS_{\epsilon, \tau}(T_i, T_j) + CATS_{\epsilon, \tau}(T_j, T_i)}{2}$. The parameters for DBSCAN are $eps = 500$ -meters and $minpts = 3$. Figure 4.15(a) shows the entropy and purity of DBSCAN, hierarchical clustering, and

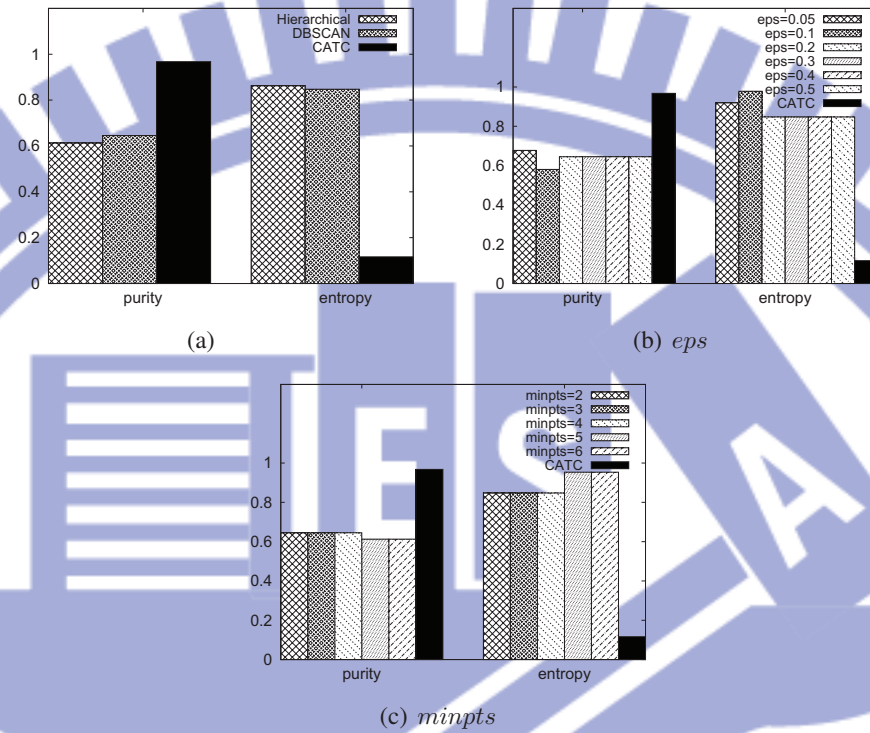


Figure 4.15: Combination of CATS and different clustering algorithms.

our proposed algorithm CATC. It can be seen in Figure 4.15(a) that CATC (our proposed clustering algorithm) have the highest purity and the lowest entropy, showing the advantage of CATC for clustering trajectories with silent durations. The reason for the bad performance of DBSCAN and hierarchical clustering is that these two traditional clustering algorithm tend to form large clusters having both Type 1 and Type 2 trajectories since these two types of trajectories have many close neighboring regions. To further investigate the impact of parameter settings (i.e., eps and $minpts$) for DBSCAN, experiments of varying these two parameters are conducted. Figure 4.15(b) and Figure 4.15(c) show that DBSCAN with all settings of parameters could not have a notable improvement in both purity and entropy. The above experiments show that our proposed clustering algorithm

will form some core sets and based on these core sets, a benefit function is used to evaluate merging operations. The design features lead to good performance of CATC compared to traditional clustering algorithms.

Next, we compare our clustering algorithm CATC with existing trajectory clustering algorithms. For purpose of comparison, two trajectory clustering algorithms are implemented: PISA [45], and TraClus [38]. PISA uses its proposed distance function and then applies OPTICS (an self-tuning DBSCAN) to cluster trajectories. TraClus involves discovering clusters of sub-trajectories, that is, sub-trajectories with similar movement behaviors are clustered, which does not reflect the purpose of our work. Thus, the TrajClus algorithm is re-designed: Supposing that the the set of all sub-trajectory clusters is expressed by $\{sc_1, sc_2, \dots, sc_n\}$, this set could be viewed as trajectory features. Thus, each trajectory T_i could be represented as a vector $\vec{v}_i = \langle x_1, \dots, x_n \rangle$, where $x_j = 1$ if T_i passes sc_j . Given a set of vectors, exploiting the *cosine similarity* to measure how similar two vectors are is quiet common. Then, DBSCAN is used to cluster trajectories. All clustering algorithms are compared by the most favorable results.

Figure 4.16(a) shows that CATC outperforms the other approaches. In PISA, trajectories are viewed as piece-wise lines and the distance function is to compare the average of the sum of Euclidean distance in each time slot. This distance function may not be appropriate for computing the distance value between trajectories with silent durations. On the other hand, TraClus could derive several common sub-trajectory clusters since trajectories of all types have several common areas. By representing trajectories into vectors of sub-trajectory clusters, the cosine similarity values of same-typed trajectories and different-typed trajectories could be similar. Thus, trajectories with different types may not easily be distinguished. From Figure 4.16(b) and Figure 4.16(c) show the entropy and purity values of all approaches. From these two figures, it could be seen that CATC still

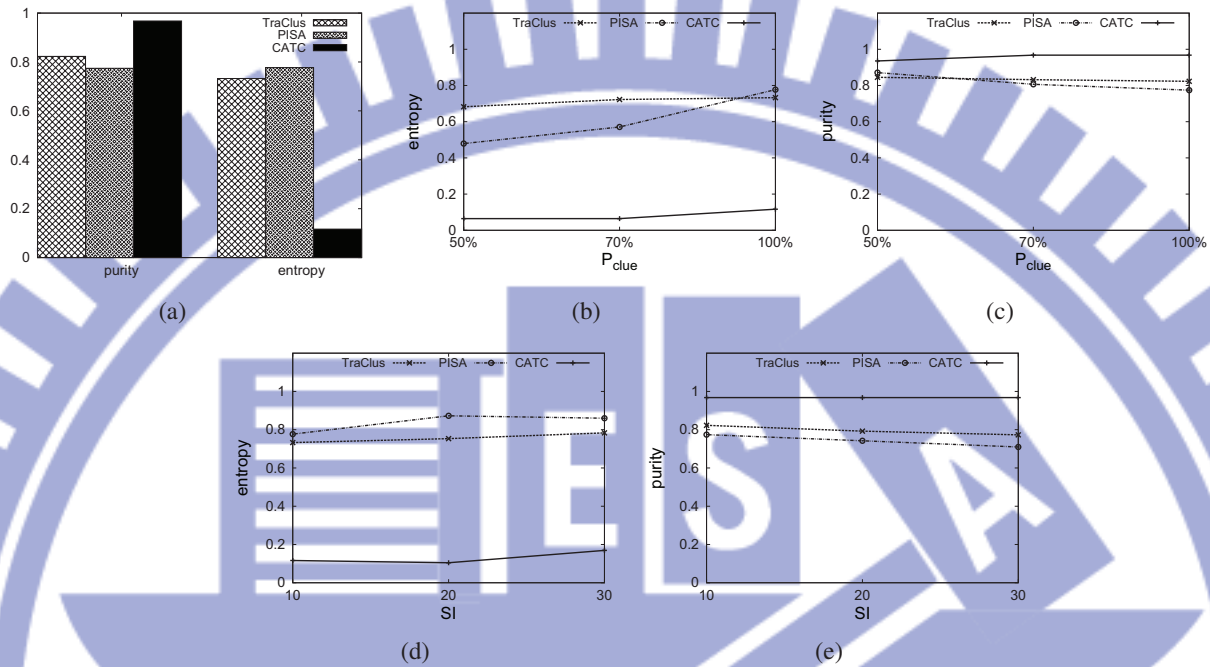


Figure 4.16: Comparison of trajectory clustering algorithms.

outperform the other approaches when the distributions of silent durations vary. While silent durations are evenly distributed, Figure 4.16(d) and Figure 4.16(e) shows that the entropy and purity values of all approaches change slightly only. Conversely, when the silent duration are randomly separated, Figure 4.16(d) and Figure 4.16(e) shows that the entropy of each approach decreases when P_{clue} decreases. Since there are several overlapping regions passed by different-typed, decreasing P_{clue} may disrupt some distance functions to distinguish the closeness of trajectories.

4.7.4 The Impact of Silent Durations for Trajectory Pattern Mining

In this experiment, we study the impact of silent durations for trajectory pattern mining. Since previous works do not have the ability to identify different kinds

of movement behavior, only one type trajectories will be investigated. We compare our proposed CACT with existing trajectory pattern mining works Spatio-temporal Frequent Pattern mining (SFP) [5], and Trajectory Pattern Mining (TPM) [18]. Note that SFP and our CACT have similar hot regions consisting representative line segments and TPM is the state-of-the-art algorithm for trajectory pattern mining.

Because our trajectory datasets contain silent durations, to compare mining results of SFP, TPM, and CACT fairly, linear interpolation and cubic spline interpolation are conducted to estimate missing points for SFP and TPM. Linear and cubic splines pass through these points with piecewise linear and cubic polynomials, respectively. For interpolation, low-order polynomials are usually used because they can reduce not only the computational costs but also the numerical instabilities that arise with higher degree curves. As reported in [58], cubic polynomials are most commonly used because no lower-degree polynomial allows a curve to pass through two specified endpoints, guaranteeing continuous first and second derivatives across all polynomial segments. Thus, these piecewise cubic polynomials can be connected smoothly.

To evaluate mining results of trajectory patterns, two performance metrics, *precision*, and *recall* are used. Since both SFP and CACT are able to derive a sequence of hot regions, a smaller region is more precise to indicate movement behavior. To quantify the precision of mining results, a number of road segments covered by hot regions is utilized. Note that trajectories are obtained during the movements of users along particular road segments. Thus, trajectories can be represented as sequences of road segments. Utilizing frequent itemset mining, a set of frequent road segments, denoted as F can then be found. Consequently, the *precision* is defined. Let C be the road segments covered by the derived regions and F be the frequent road segments derived by frequent itemset mining.

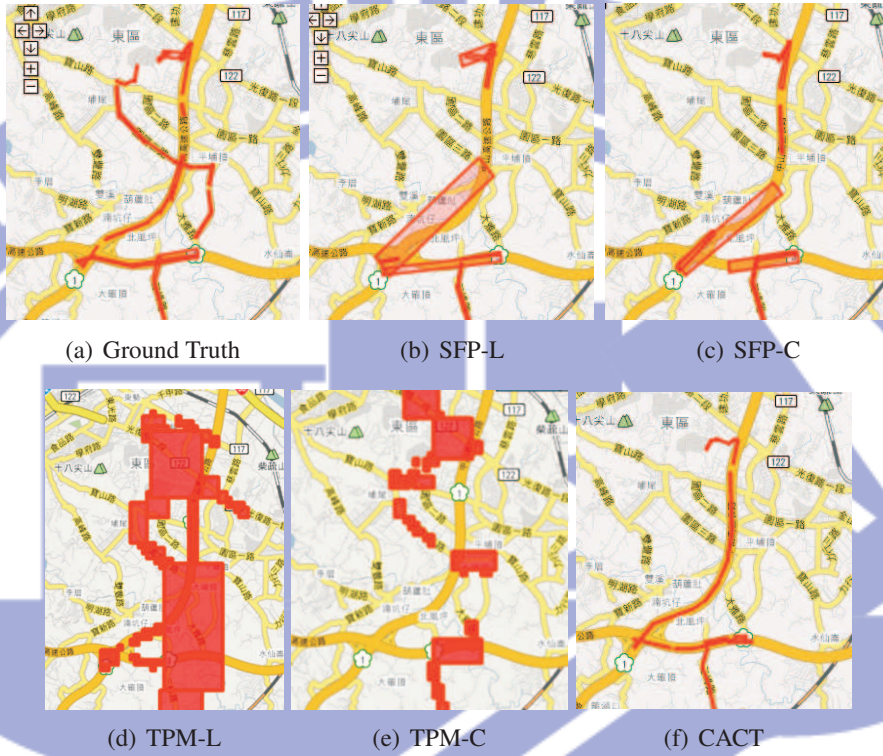


Figure 4.17: Comparison of TPM, SPF, and CACT.

The *precision* is formulated as $L(C \cap F)/(L(C \cap F) + L(C \cap \bar{F}))$, where $L(\cdot)$ represents the total length of the roads. A higher precision value means that the derived region tends to cover more frequent road segments. By contrast, the *recall* involves evaluating the number of frequent road segments within the derived hot regions. As such, the *recall* is formulated as $L(C \cap F)/(L(C \cap F) + L(\bar{C} \cap F))$. A higher recall value means that more frequent road segments can be covered by the derived regions.

First, we compare trajectory patterns derived by SFP, TPM, and CACT in a visualized manner. Figure 4.17(a) shows the movement behavior (i.e., Type 1 movement behavior). By setting $P_{clue} = 50\%$, Type 1 trajectories may have more silent durations. Then, given the trajectories with silent durations, trajectory pat-

terns derived by all approaches are shown in Figure 4.17. Figure 4.17(b) shows trajectory patterns derived by SFP-L. Silent durations fragment trajectories such that trajectory patterns of SFP-L have larger hot regions. Even though the cubic spline interpolation attempts to form trajectories more smoothly, trajectory patterns mined by SFP-C are of higher quality than those of SFP-L. From Figure 4.17(b) and Figure 4.17(c), some regions are not found, as compared to the baseline in Figure 4.17(a). Conversely, Figure 4.17(d) and Figure 4.17(e) show trajectory patterns of TPM by given trajectories with linear and cubic spline interpolations. TPM uses a set of neighboring grids as hot regions. To "smooth" the hot regions, linear regression is also used further to connect two nearby hot regions. Due to this mechanism, Figure 4.17(d) shows that TPM may generate many large hot regions when trajectories are linearly-interpolated. Some hot regions cover some areas in which a user will never appear. On the other hand, when trajectories are smoothed by cubic spline interpolation, Figure 4.17(e) shows that there are more hot regions located in the roads in the previous one. Compared with the baseline, this trajectory pattern cannot describe the movement of this user effectively. Finally, Figure 4.17(f) shows the trajectory patterns mined by CACT. Compared with the baseline in Figure 4.17(a), CACT could derive trajectory patterns that are almost the same as the baseline. Some less frequent hot regions cannot be discovered when trajectories have certain silent durations.

Next, we evaluate the precision and recall of these approaches. Figure 4.18(a) shows that CACT leads to the highest precision among all approaches. The precision of CACT is larger than 70% in all cases. Because hot regions of SFP are rectangular, SFP can achieve higher precision than TMP. Figure 4.18(b) shows the recall values of all approaches. CACT outperforms other approaches as well. Since hot regions of TMP are much larger than that of SFP, they could cover a lot of road segments. Thus, the recall of TMP could be higher than that of SFP.

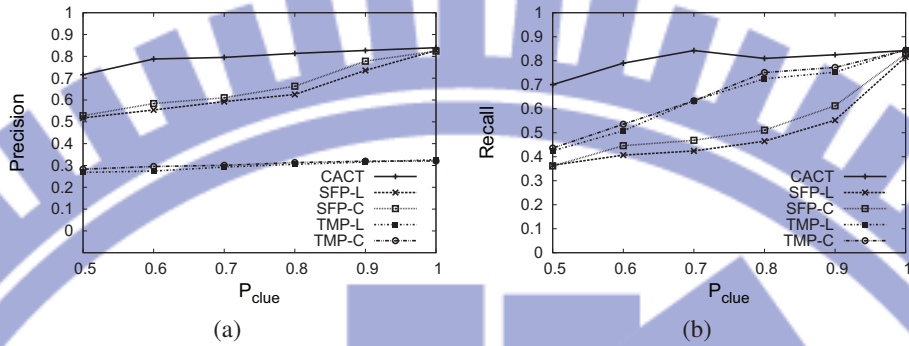


Figure 4.18: Precision and recall of SFP-L, SFP-C, TMP-L, TMP-C and CACT with P_{clue} varied.

	CACT	SFP-L	SFP-C	TPM-L	TPM-C
CarWeb	7.24	2.12	2.23	1.42	1.57
Short Synthetic Dataset	27.4	5.31	5.51	4.76	4.82
Long Synthetic Dataset	32.3	5.84	6.23	5.22	5.23

Table 4.3: Execution Time (in seconds).

To conclude, CACT could handle silent duration very well when $P_{clue} > 50\%$. for the other approaches, P_{clue} is the most critical factor for increasing precision, whereas using interpolation on trajectories cannot increase precision considerably.

CACT may need longer execution time than SFP and TPM. Table 4.3 discusses the execution time of all approaches. Three datasets are used in this experiment, including CarWeb dataset, short synthetic dataset (average length of each trajectory is 1000) with 100 trajectories and long synthetic dataset (average length of each trajectory is 4000) with 100 trajectories. The execution time of CACT needs at least three times than the other approaches. The main reason is that CACT first clusters trajectories and then generates trajectory patterns, whereas the other approaches do not cluster trajectories first. However, this tradeoff could make a significant improvement in precision and recall of trajectory patterns.

4.7.5 Sensitivity Analysis for CACT

This section discusses the sensitivity analysis of both algorithm parameters (ϵ , τ , min_sup , λ) and environment parameters (SI , P_{clue}). Following the previous settings, the default values of parameters are $\lambda = 0.3$, $\tau = 300$ seconds, $\epsilon = 100$ meters, $min_sup = 0.1$, $r = 300$ meters, $t = 300$ seconds, $P_{bias} = 80\%$, and $P_{clue} = 50\%$.

Impact of Thresholds on Similarity Measurements

In this section, we investigate the impact of environment variables to the similarity measurements. Here, environmental variables include P_{clue} , r , and t . To compare how many value of a distance function is incurred by the environmental variable, the normalized distance is used as a metric which is defined as follows: given a distance function d , the normalized distance for the distance value $d(x)$ is $\frac{d(x)}{MAX-min}$, where the minimum value of d is min and the maximum value of d is MAX . This measure is used to test sensitivity of the distance function with respect to some environment variables. In the following experiments, a trajectory is randomly selected from our real dataset and a synthetic trajectory is generated by applying the environmental variables. Then, the normalized distance between these two trajectories are computed. Finally, all results presented is the average of the normalized distances for 20 times.

Figure 4.19(a) discusses the normalized distance with P_{clue} varied. It could be seen that with the decrease of P_{clue} , the normalized distance of EDR increases the most significantly, that of CATS increases less significantly, and that of LCSS does not change. The reason is that LCSS only considers how many parts of two trajectories are matched. Moreover, LCSS is normalized by the length of the shortest trajectory such that the normalized distance keeps zero. On the other hand, EDR not only considers the matched parts but also gives penalty to gaps

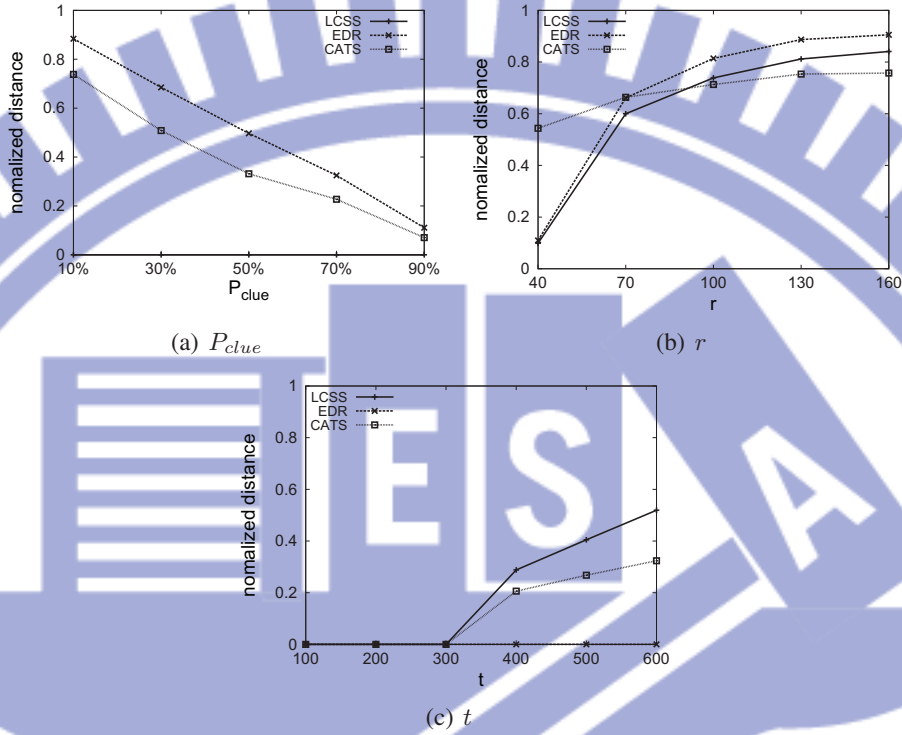


Figure 4.19: Normalized distance of LCSS, EDR, and CATS with P_{clue} , r , and t varied.

between two trajectories. When P_{clue} decreases, EDR gives more penalty to these two trajectories. Note that LCSS is not suitable if trajectories contain silent durations since it is not sensitive to P_{clue} . On the other hand, EDR emphasizes giving gap penalty rather than reflects the amount of the common matched parts. To reflect the amount of clues, CATS uses a continuous quantization to capture the common matched parts and is normalized by the length of the trajectory itself. Thus, CATS is sensitive to P_{clue} but CATS does not emphasizes the gaps too much. Figure 4.19(b) shows the impact of spatial shifting r . Note that $r = 10$ meters is decided by the average error of GPS. When r increases, three distance functions keep growing significantly in the beginning but slowly in the end. The reason is that points can only be matched within the range of $\epsilon = 10$ meters. Thus,

the normalized distances almost keep constant when r exceeds 10 meters. This experiment shows that these three distance functions are almost sensitive to the spatial bias r . Figure 4.19(c) shows the impact of time bias t . EDR keeps constant because it does not take time into account. LCSS and CATS keeps growing with t increasing because a point is hard to find the matched point. To conclude, CATS strikes a compromise on LCSS and EDR, which can fit the scenario that trajectories exhibit certain silent durations.

The Impact of Threshold on Clustering Algorithms

In this section, we discuss the impact of the threshold λ used in our clustering algorithm CATC. This threshold is used to identify whether two trajectories have enough strong clues. Figure 4.20 shows purity and entropy with λ varied. Figure 4.20(a) shows that the purity of CATC is not sensitive to the change of λ . We observed that the number of clusters becomes larger when λ increases. For example, the number of clusters is 4 when $\lambda = 0.2$, whereas the number of clusters is 6 when $\lambda = 0.4$. The reason is that when λ increase, a clue-graph becomes sparser such that a larger cluster with the same label may be separated into two smaller clusters with the same label. Therefore, the purity value does not decrease. Figure 4.20(a) shows the entropy of CATC slightly decrease when λ increase. When λ is small, a clue-graph becomes sparser. Trajectories of Type 1 and Type 2 are easily clustered into the same cluster since they pass some nearby regions. In this experiment, $\lambda = 0.2$ could achieve the best balance. Thus, this value is as our default setting. Generally speaking, the value of λ is highly dependent on the distribution of data. Choosing a larger λ may lead to higher purity. The number of trajectories in a cluster may decrease, which may have not enough trajectories to derive trajectory patterns. On the other hand, a smaller λ may lead to higher entropy. The number of trajectories in a cluster may not belong to the same movement behavior,

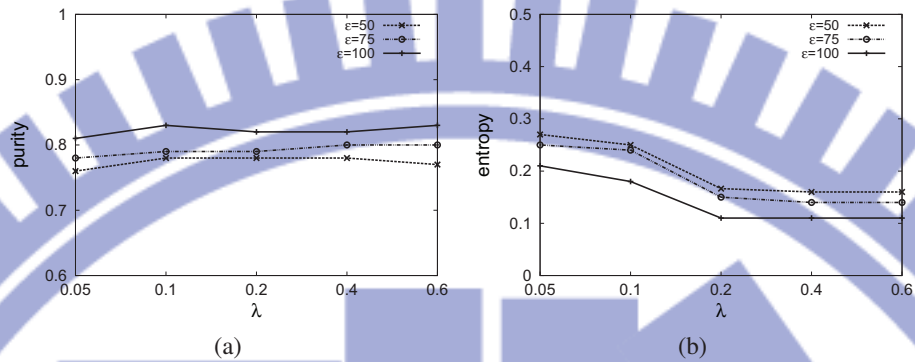


Figure 4.20: Purity and Entropy of CACT with λ varied.

which may derive improper hot regions.

The Impact of Thresholds on Trajectory Pattern Mining

In CACT, the spatial-bias threshold ϵ is a user-specified threshold in the spatial decaying function. The temporal-bias threshold τ should be given in CACT as tolerating the time bias. Now, we conduct experiments to investigate the impact of two thresholds on CACT.

Figure 4.21 shows the precision and recall with ϵ varied. Let the unit of r be one meter. The precision and recall of $r = 300$ are almost constant in all cases of ϵ . On the other hand, the precision and recall of $r = 200$ and $r = 100$ keep growing when ϵ increases. In general, the precision and recall keep growing until the ϵ is larger than r . When the ϵ is larger than the spatial bias r , the size of derived regions is allowed to contain the range of the spatial bias generated in the synthetic trajectories. In other words, a larger ϵ can tolerate trajectories with a larger spatial bias. Clearly, the precision and recall can be effectively improved.

On the other hand, the threshold of τ is used to tolerate the existence of a certain temporal bias between trajectories. Figure 4.22 shows the precision and recall with τ varied. Let the unit of t be one second. Similarly, it can be seen that

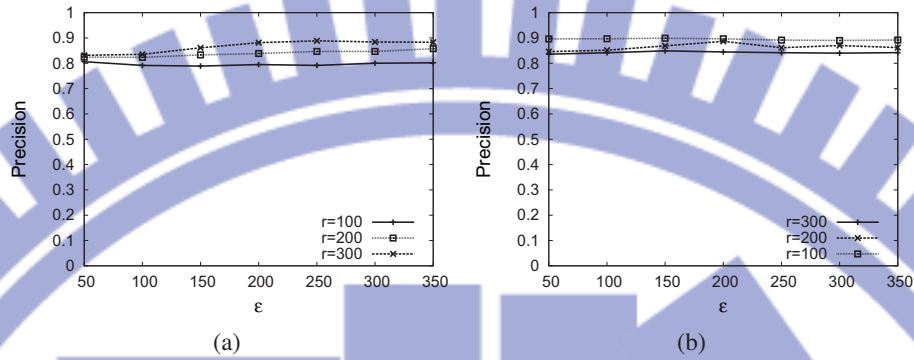


Figure 4.21: Precision and recall of CACT with ϵ varied.

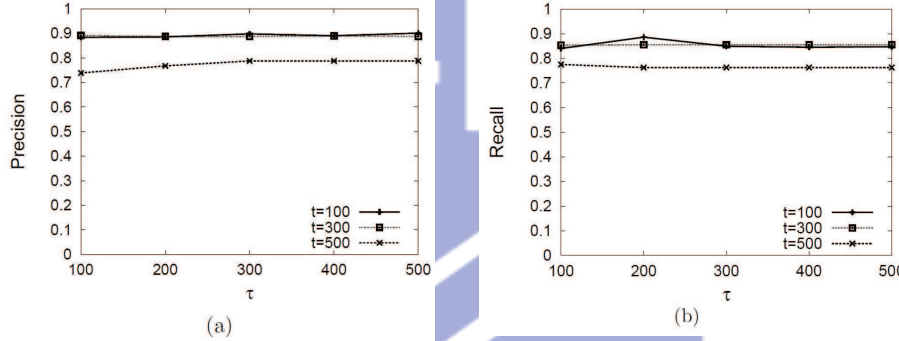


Figure 4.22: Precision and recall of CACT with τ varied.

the precision and recall keeps growing when the t is smaller than τ . Moreover, the precision and recall are less sensitive when temporal bias t varied than spatial bias r varied. It is because that various setting of τ easily affects the number of clusters. Note that the trajectories in the synthetic dataset follow four kinds of movement behaviors. We observe that the number of clusters becomes 6 if $t \leq \tau$, where two clusters with two movement behaviors are divided into four smaller clusters with two movement behaviors. It will not affect the precision and recall because each two smaller clusters are conducted by trajectories that have the same movement behavior.

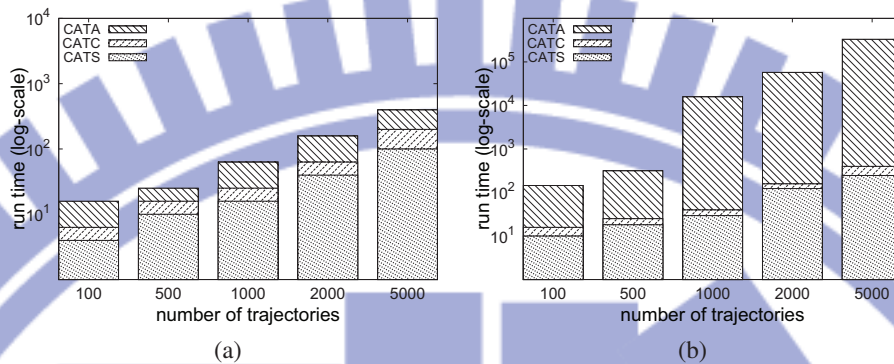


Figure 4.23: Execution time under (a) short and (b) long trajectories.

Execution Time

The execution time with the number of trajectories varied are discussed here. Two kinds of synthetic trajectories are used where each trajectory has about 2000 points in short trajectories, and has 6000 points in long trajectories. The execution time includes three parts: 1. computing similarities between trajectories (CATS), 2. clustering trajectories (CATC), and 3. generating trajectory patterns (CATA). Figure 4.23 shows the experimental results which the execution time is represented by log-scale. Overall, the execution time increases with the increasing of the number of trajectories. However, the proportion of the clustering phase and the aggregation phase are not the same in different kinds of trajectories. In the short trajectories, as shown in Figure 4.23(a), the clustering phase takes more execution time than the aggregation phase. On the contrary, Figure 4.23(b) shows that it spends most execution time to derive hot regions when long trajectories are considered. Longer trajectories needs more time for aggregating the information and executing Douglas-Peucker line simplifier.

4.8 Conclusions

In this paper, we proposed algorithm CACT to discover trajectory patterns. Since users may have multiple movement behaviors, we claimed that trajectories should be clustered before the determination of hot regions for trajectory pattern mining. However, trajectories usually have both the spatial and temporal bias. Furthermore, in this paper, we further claimed that trajectories usually have silent durations. During silent durations, no detailed movements will be presented. Previously developed methods for clustering trajectories cannot be directly applied. Note that trajectories that represent the same movement behavior usually have some clues. Clues of trajectories reflect some common spatial regions appeared in trajectories. Thus, we formulate a clue-aware trajectory similarity. Based on this similarity between two trajectories, in CACT, we developed a clue-aware clustering approach in which trajectories are clustered according to clues hidden in trajectories. Once the groups of trajectories are determined, CACT will aggregate trajectories in the same group to identify hot regions of trajectory patterns. Our evaluation of CACT, using both real and synthetic datasets in our experiments. Our proposed algorithms are compared with three kinds of research works (i.e., similarity measurements, clustering and trajectory pattern mining). Experimental results show that CACT is able to effectively discover trajectory patterns even if trajectories only capture fragments of movement behaviors. For the further works, the efficiency of CACT could be enhanced for large-scale trajectory datasets. In addition, for the clue-aware trajectory aggregation, more efficiency algorithms should be designed.



Chapter 5

Exploiting Trajectory Profiles for Mining User Communities

5.1 Introduction

Web-based social networks have attracted more and more research efforts in recent years. In particular, community mining is one of the major directions in social network analysis where a community can be simply defined as a group of objects sharing some common properties. Nowadays, with the rapid development of positioning techniques (e.g., GPS), one can easily collect his/her trajectories. Many GPS community sites are then established and users are easily share their trajectories [1][3]. Furthermore, with a large amount of trajectories shared, users expect to have one trajectory search or recommendation to rank these trajectories interested. For example, one would like to find some friends who have the same traveling interests, or one may want to know some interesting traveling paths different from his habits. As such, mining communities among users is helpful for trajectory recommendation or search.

In this paper, we target at the problem of mining user communities from users' trajectories, where a community refers to a set of users who have similar moving behaviors. Prior works have elaborated on mining trajectory patterns that cap-

ture moving behaviors[29][18][5]. Note that trajectory patterns usually have a huge number of patterns and these patterns are not easily to formulate distance measurements of users. In other words, trajectory patterns mined are not well-organized for mining communities, let alone deriving the distance measurement of trajectory patterns. Therefore, to find the community in a location-based social network, one important issue is to build a *trajectory profile* for a user, where a trajectory profile is referred as a data structure which can organize trajectory patterns for a user.

To address the issues above, in this paper, we adopt a probabilistic suffix tree (abbreviated as PST) to represent a trajectory profile of a user [48][61][49]. A PST is shown to be able to accurately capture user moving behavior and represents the trajectory patterns of a user into a tree structure. In light of the proposed trajectory profiles, the distance measurement is formulated for discovering communities. Explicitly, our design of mining communities consists of three steps: 1.) constructing trajectory profiles of users, 2.) formulating distance measurements among trajectory profiles and 3.) clustering similar trajectory profiles of users into groups. Since new users may be added, to efficiently identify which community to include these new users, for each community, we further select one representative PST. By comparing similarities with representative PSTs, a new user is able to quickly find which community that he/her should join. To evaluate the performance of the proposed approaches, we conduct comprehensive experiments and experimental results show that the trajectory profile proposed is able to accurately reflect user moving behaviors of users and is very helpful to mine communities of users. From our experiments, communities mined are indeed referred those users who have similar moving behaviors.

The contributions of this paper are summarized as follows:

1. A trajectory profile, represented as a probability suffix tree (PST), is first

proposed. Trajectory profiles proposed can not only capture moving behaviors but also are utilized for discovering communities.

2. A distance measurement between two PSTs is derived. By exploring the concept of editing distances, the distance measurement can truly reflect how closeness between two PSTs.

3. According to the distance among PSTs, one clustering algorithm is developed to identify communities in which similar users are put together.

4. To efficiently identify which community for a new user, a representative PST of a community is determined.

The rest of the paper is organized as follows. In Section 4.2, our approach for mining community in a location-based social network is presented. Experimental results are shown in Section 4.3. Section 4.4 concludes with this paper.

5.2 The Framework of Mining Communities

In this section, we develop a framework to discover user communities. The design of our framework consists of three steps:

Step 1. Constructing Profiles: In this step, frequent regions of trajectories are derived. Based on these frequent regions, we propose a probability suffix tree (abbreviated as PST) as trajectory profiles for users.

Step 2. Formulating Distance of Profiles: In this step, we derive the distance measurement of users in terms of their profiles (i.e., their PSTs).

Step 3. Identifying Community: According to distance values derived in Step 2, we develop a clustering algorithm to identify communities of users, where users have similar trajectory profiles are in the same community.

After generating communities of users, for each community, we judiciously select one representative PST tree. As pointed our early, for a new user, we only

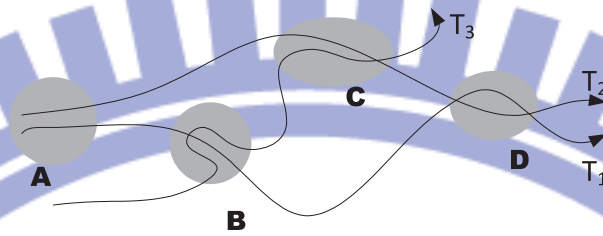


Figure 5.1: Frequent regions by the density-based approach.

need to compare his/her trajectory profiles with representative PSTs. The selection of representation PSTs for each group is finally presented in Section 3.4.

5.2.1 Constructing Profiles

In this phase, by given trajectories of users, we would like to construct their profiles (i.e., PSTs). By given all trajectories of users, as the same as ordinary trajectory pattern mining approaches, the frequent regions of trajectories are first derived. Then, by representing each trajectory into a sequence of frequent regions, a PST is then constructed for each user.

Many previous works for determining frequent regions in trajectory patterns adopts the density-based approach [29][18]. In the density-based approach, a region is viewed as a frequent region if the number of trajectories passing by is larger than a pre-defined threshold. Furthermore, if nearby regions are also frequent regions, these regions could merge into one larger region. For example, given three trajectories in Figure 5.1, four frequent regions A, B, C, and D are derived by the density-based approach in [30].

After deriving the frequent regions from trajectories of all users, trajectories can be transformed into a sequence of frequent region ids. For example, T_1 in Figure 5.1 can be represented as $\langle A, B, D \rangle$. Such transformation can guarantee that the noise does not affect the procedure of trajectory pattern mining. Once

transforming trajectories of each user, we can adopt a probabilistic suffix tree to capture moving behavior of each user. Specifically, each edge of a PST is labeled by a frequent region id that indicate one movement from one frequent region to the other one of a user. Each tree node is labeled by a string that shows a path from the node to the root. In other words, a *tree node* is labeled as $r_k \dots r_2 r_1$ can be reached from the traversal path from $root \rightarrow r_1 \rightarrow r_2 \rightarrow \dots \rightarrow r_k$. In a PST, each tree node maintains a *conditional table* to record the appearing counts and the conditional probabilities of next frequent regions that follow the label of the tree node. For example, in Figure 5.2, the conditional table of tree node "AB" in T_1 shows that the conditional probability of next frequent region "A" after "AB" is 1. Clearly, tree nodes' labels show the frequently moving regions of a user and the corresponding conditional tables are used for predicting the next movements.

The construction of a PST is briefly described as follows. At the beginning, the PST has only one root node with the counts of each moving record appearing in the buffer so far. If the count of moving record r_i is larger than the predefined threshold (i.e., minimal support denoted as $MinSup$), one tree node labeled as r_i will be created as the child node of the root. Similarly, tree node r_i will maintain the occurrence count of r_i and the probability distribution table is also associated with the node to record the conditional probability of the next movement with the prefix segment r_i . Assume that the movement held by the buffer are $r_1 \dots r_{\ell-1}$. When a new movement, r_ℓ arrives into the buffer, those statistical information (i.e., counts and the conditional probabilities) should be updated accordingly. Interested readers are referred to [49] for the detailed procedure of constructing a PST.

5.2.2 Formulating Distance of Profiles

As mentioned above, a PST is a profile which represent trajectory patterns of each user. In this phase, to determine distance of users in terms of their profile, we formulate a distance function to measure the distance of any two users in terms of their profiles.

To measure the distance between two PSTs, both the structure and the statistical information of a PST should be considered. Since the label of a node in a PST is the suffix of the label of its child nodes, a branch in a PST represents a series of frequent sequential patterns (represented as sequences of sensor ids). In other words, by extracting a branch in a PST, frequent sequential patterns with the same destination are determined. For example, the second branch in T_1 in Figure 5.2 is $root \rightarrow B \rightarrow AB$, which represents that the frequent sequential patterns with the destination as B are B and AB . Such frequent sequential patterns extracted from branches of a PST are referred to as structure information in a PST. As a result, the similarity of two PSTs should take the structure information of PSTs into consideration. For example, consider three PSTs T_1 , T_2 and T_3 in Figure 5.2 as our example. It can be verified that the structure of T_1 is more similar to that of T_3 because T_1 and T_3 have the more common branches (i.e., $root \rightarrow A$ and $root \rightarrow B \rightarrow AB$) than T_2 does. Except the structure information of PSTs, we should further consider the statistical information of PSTs. Note that two users may have similar moving behaviors in terms of sequential patterns. However, their behaviors are different if the probabilities of these sequence patterns is considered. For example, in Figure 5.2, these two PSTs have the same branches but their probabilities are different. The left PST indicates that one user frequently stays A, whereas the right PST shows that one user frequently appears in area B. Therefore, the probabilities of nodes in a PST should also be considered. Through

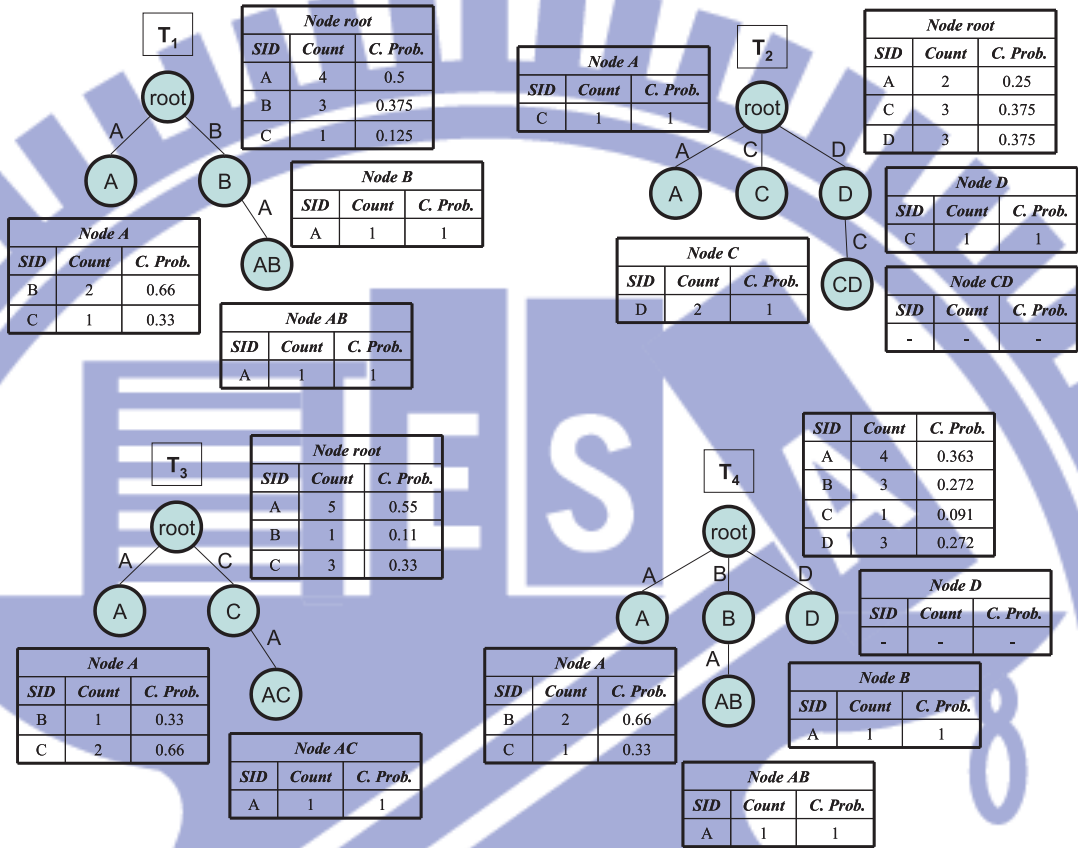


Figure 5.2: An example profile of PSTs.

statistical information in PSTs, each node can determine its corresponding importance in terms of the probability that indicates how frequent one user travels along this frequent sequential pattern. Thus, when designing a distance function among PSTs, we should give higher weight to a node with higher probability. For example, consider a PST T_2 in Figure 5.2, we could obtain that the probability that the user stays in area C is 0.33. On the other hand, the probability that the user stays in area A and then stay in area C is $0.55 \times 0.66 \approx 0.36$ (i.e., $P(AC) = P(A) \times P(C|A)$). Thus, node AC will be given a higher weight than C when we design a distance function.

By exploiting PST tree structures and statistical information in conditional

tables, a distance function δ_{MSL} is formulated. In order to capture structure similarity of PSTs, we transform a PST into a *moving sequence list*, in which each element is a moving sequence from the root node to a leaf node and elements are ordered from the left to the right of PSTs. A moving sequence of a PST is defined as follows:

Definition. Moving Sequence: Given a PST T_i , the j -th moving sequence is defined as $L_i^j = [TL_{i,j}^1:p(TL_{i,j}^1), TL_{i,j}^2:p(TL_{i,j}^2), \dots, TL_{i,j}^\ell:p(TL_{i,j}^\ell)]$, where $TL_{i,j}^k$ denotes the k -th tree node traversing from the root in the j -th branch of the root node and $p(TL_{i,j}^k)$ is the corresponding probability.

For example, consider the second branch of a PST T_1 in Figure 5.2. Since the second branch of the root is $B \rightarrow AB$, we can obtain $L_1^2 = [B : 0.375, AB : 0.33]$. To measure the importance of a moving sequence, the weight of each moving sequence is defined as follows:

Definition. Weight of a Moving Sequence: The weight of a moving sequence L_i^j is formulated as $w(L_i^j) = \sum_{k=1}^n p(TL_{i,j}^k)$, where n is the number of elements in L_i^j .

For example, the weight of a moving sequence $[A : 0.5]$ is 0.5 and the weight of moving sequence $[B : 0.375, AB : 0.33]$ is 0.705 (i.e., $0.375 + 0.33$). Consequently, a moving sequence list is defined as follows:

Definition. Moving Sequence List: Given a PST T_i , the moving sequence list is defined as $L_i = \langle L_i^1, L_i^2, \dots, L_i^n \rangle$, where n denotes the number of moving sequences and L_i^j is the j -th moving sequence.

For example, the moving sequence list in T_1 is $\langle [A : 0.5], [B : 0.375, AB : 0.33] \rangle$. Consequently, a moving sequence list derived from a PST is able to represent both the structure and the statistical information of a PST. As such, we propose a distance function δ_{MSL} . Given two PSTs T_i and T_j with their moving sequence lists L_i and L_j , the distance between two PSTs $\delta_{MSL}(T_i, T_j)$ is determined by the editing distance between L_i and L_j . The editing distance between L_i and L_j is determined as the minimal *cost* of transforming L_i into L_j via three editing operations (i.e., *insertion*, *deletion* and *replacement*). To facilitate the presentation, let $TL_{i,m}^\ell$ and $TL_{j,n}^\ell$ be a pair, $L_i^j[1..n]$ be L_i^j with n elements, and $L_i[1..m]$ be L_i with m moving sequences. Three operations and the corresponding costs are described as follows:

Insertion: We transform $L_i[1..m]$ to $L_j[1..n]$ by (1) transforming $L_i[1..m]$ to $L_j[1..n-1]$ and then (2) *inserting* one moving sequence L_j^n into $L_j[1..n-1]$. The corresponding cost is the sum of the cost of transforming $L_i[1..m]$ to $L_j[1..n-1]$ and $w(L_j^n)$.

Deletion: We transform $L_i[1..m]$ to $L_j[1..n]$ by (1) transforming $L_i[1..m-1]$ to $L_j[1..n]$ and (2) *deleting* one moving sequence L_i^m . The corresponding cost is the sum of the cost of transforming $L_i[1..m-1]$ to $L_j[1..n]$ and $w(L_i^m)$.

Replacement: We transform $L_i[1..m]$ to $L_j[1..n]$ by (1) transforming $L_i[1..m-1]$ to $L_j[1..n-1]$ and (2) *replacing* a moving sequence L_i^m to a moving sequence L_j^n . Specifically, to replace one moving sequence L_i^m as L_j^n , we compare nodes in two moving sequences pair by pair. If the labels of nodes in moving sequences are the same, the cost is estimated as the difference of their probability. Then, we keep comparing the next pair until the pairs of two moving sequences are not the same. The probabilities of remaining pairs are summed up as the cost. For example, suppose that two moving sequences $[C : 0.1, AC : 0.2, BAC : 0.3]$ and

$[C : 0.3, BC : 0.5]$. Since the labels of the first pair are the same (i.e., C), the cost of this replacement is $|0.1 - 0.3| = 0.2$. On the other hand, since the labels of the second pairs are not the same (i.e., AC and BC), the cost for replacing $[AC : 0.2, BAC : 0.3]$ to $[BC : 0.5]$ is $0.2 + 0.3 + 0.5 = 1$ (i.e., AC:0.2, BAC:0.3 and BC:0.5). Therefore, the total cost for this replacement is $0.2 + 1 = 1.2$. Consequently, the cost is the sum of probabilities of all elements. The corresponding cost is the sum of the cost of transforming $L_i[1..m - 1]$ to $L_j[1..n - 1]$ and the cost for replacing L_i^m to L_j^n .

According to the above three operations, we can define the editing distance between two moving sequences lists. Assume that $R(\cdot)$ denotes the cost of replacement operation.

Definition. Editing Distance: Given two moving sequence lists L_i and L_j , the editing distance is defined as follows:

$$ed(L_i[1..m], L_j[1..n]) = \min \begin{cases} ed(L_i[1..m], L_j[1..n - 1]) + w(L_j^n) \\ ed(L_i[1..m - 1], L_j[1..n]) + w(L_i^m) \\ ed(L_i[1..m - 1], L_j[1..n - 1]) + R(L_i^m, L_j^n) \end{cases}$$

$$\text{where } R(L_i^m, L_j^n) = \begin{cases} R(L_i^m[2..k], L_j^n[2..\ell]) + |p(TL_{i,m}^1) - p(TL_{j,n}^1)|, \text{ if } TL_{i,m}^1 = TL_{j,n}^1 \\ \sum_{c=1}^k p(TL_{i,m}^c) + \sum_{c=1}^{\ell} p(TL_{j,n}^c), \text{ otherwise} \end{cases}$$

The boundary conditions are as follows:

$$ed(L_i[0], L_j[1..n]) = \sum_{c=1}^n w(L_j^c),$$

$$\text{and } ed(L_i[1..m], L_j[0]) = \sum_{c=1}^m w(L_i^c).$$

Thus, based on the editing distance function, given two PSTs T_i and T_j with their moving sequence list L_i and L_j , the distance function $\delta_{MSL}(T_i, T_j)$ is defined as $ed(L_i, L_j)$.

The distance function δ_{MSL} derives the distance of two PSTs by computing the editing distance between their moving sequence lists. By exploiting dynamic programming, Algorithm DP (**D**istance of **P**STs) is proposed to compute the editing distance between two moving sequence lists from the above recurrence relation in a bottom-up manner. Without loss of generality, given two PSTs T_1 and T_2 , the table entry $c[i, j]$ represents the minimal cost of transforming $L_1[1..i]$ to $L_2[1..j]$. To compute the minimal cost in a bottom-up manner, the $c[0, j]$ and $c[i, 0]$ for all i and j are determined first and then compute the table entries row by row. For each table entry, three costs should be calculated according to three editing operations (i.e., insertion, deletion and replacement). Among three costs, each table entry only stores the minimal cost.

Algorithm 8: : Algorithm DP

Input: $L_i[1..m]$, and $L_j[1..n]$: two moving sequence lists
Output: $c[m, n]$, the distance of two moving sequence lists
1: **for** $i = 0$ to m **do**
2: $c[i, 0] \leftarrow \sum_{c=1}^m w(L_i^c)$;
3: **for** $j = 0$ to n **do**
4: $c[0, j] \leftarrow \sum_{c=1}^n w(L_j^c)$;
5: **for** $i = 1$ to m **do**
6: **for** $j = 1$ to n **do**
7: $x = c[i, j - 1] + w(L_j^n)$;
8: $y = c[i - 1, j] + w(L_i^m)$;
9: $z = c[i - 1, j - 1] + R(L_i^m, L_j^n)$;
10: $c[i, j] \leftarrow \min(x, y, z)$;
11: **end for**
12: **end for**

For example, the distance between T_1 and T_3 can be determined by calculating $d(L_1[1..2], L_3[1..2])$. Note that our goal is to derive the minimal cost to transform $L_1[1..2]$ into $L_3[1..2]$ such that the value in $c[2, 2]$ is the distance $\delta_{MSL}(T_i, T_j)$. Table 5.3 shows the execution scenario for Algorithm DP. By taking $c[1, 2]$ as our example, we compare the costs of transforming $\langle [A : 0.5] \rangle$ to $\langle [A : 0.55], [C : 0.33, AC : 0.36] \rangle$ by insertion, deletion and replacement. Among these three

costs, only the minimal one is kept in $c[1, 2]$. Explicitly, the cost for *insertion* is the sum of (1) $c[1, 1] = 0.05$: the cost transforming $\langle [A : 0.5] \rangle$ to $\langle [A : 0.55] \rangle$, and (2) the weight of L_2^2 : 0.69 (i.e., $0.33 + 0.36$). Thus, the total cost is 0.74. Then, the cost for *deletion* is the sum of (1) $c[0, 2] = 1.24$: the cost transforming an empty moving sequence $\langle [] \rangle$ to $\langle [A : 0.55], [C : 0.33], [AC : 0.36] \rangle$, and (2) the weight of L_1^1 : 0.5 (i.e., $0.33 + 0.36$). Thus, the total cost is 1.74. Finally, the cost for *replacement* is the sum of (1) $c[0, 1] = 0.55$: the cost transforming an empty moving sequence $\langle [] \rangle$ to $\langle [A : 0.55] \rangle$, and (2) the cost of replacement of $[A : 0.5]$ and $[C : 0.33, AC : 0.36]$: 1.19 (i.e., $0.5 + 0.33 + 0.36$). Thus, the total cost is 1.74. By taking the minimal cost between 0.74, 1.74 and 1.74, we can obtain the value of $c[1, 2]$ is 0.74. Following the same process, we can derive that $c[2, 2]$ is 1.445. Therefore, $\delta_{MSL}(T_1, T_3) = 1.445$.

Consider three PSTs T_1, T_3 and T_4 in Figure 5.2 as our example. We derive that $\delta_{MSL}(T_1, T_3)$ is 1.445 and $\delta_{MSL}(T_1, T_4)$ is 0.602. By comparing tree structures of these three PSTs, it can be verified that T_1 is more similar to T_4 than T_3 since both T_1 and T_4 have sequences A and $A \rightarrow B$ where users frequently travel. However, T_1 and T_3 have only sequence A . Thus, T_1 is more similar to T_4 , which is validated by $\delta_{MSL}(T_1, T_4) < \delta_{MSL}(T_1, T_3)$.

5.2.3 Identifying Community

Once deriving the distance between profiles, the following task is to identify communities of users. The community identification problem can be formulated as follows:

Definition. Community Identification Problem: Let the distance function for PSTs be $\delta_{MSL}(\cdot)$. Given a set of users $\{U_1, \dots, U_n\}$ with their profiles $\{T_1, \dots, T_n\}$, the distance threshold δ , divide users into communities such that the number of communities is minimal and each U_i, U_j in the same community should satisfy

	[] 0	[(A:0.55)] 1	[(C:0.33), (AC:0.36)] 2
[] 0	<u>0</u>	<u>0.55</u>	<u>1.24</u>
[(A:0.5)] 1	<u>0.5</u>	1.05 (i) 1.05 (d) <u>0.05 (r)</u>	<u>0.74 (i)</u> 1.74 (d) 1.74 (r)
[(B:0.375), (AB:0.33)] 2	<u>1.205</u>	1.775 (i) <u>0.775 (d)</u> 1.775 (r)	<u>1.445 (i)</u> 1.445 (d) 1.445 (r)

Figure 5.3: An execution scenario for the distance of T_1 and T_3 . Only underlined values will be stored in this table.

$$\delta(T_i, T_j) \leq \delta.$$

To solve this problem, a graph can be first constructed where each vertex represents a user and there is an edge between two vertices if the distance of two PSTs are smaller than δ . Obviously, the community identification problem can be modeled as a minimal clique problem, where a graph is required to be decomposed into the minimal number of cliques. Algorithm CI (Community Identification) is then proposed. At first, vertices with larger degrees are selected to obtain larger cliques. Thus, from line 2 to line 3, we start to select vertex T_i with the highest degree in graph G . In line 4, those vertices adjacent to vertex T_i are put into list L . Then, the PSTs in list L form a graph together and recompute their node degrees (line 5). This step only calculates the degrees of vertices in list L . We construct a group C_i that contains the PST T_i and repeatedly select the PST T_j with the highest node degree in list L . Moreover, the PST T_j is included in group C_i if the PST T_j has edges with all PSTs in C_i (from line 7 to line 12). Finally, PSTs in group C_i are removed from graph G (line 14). Following the above operations, we could discover all cliques until all vertices are visited (i.e., G is empty).

Algorithm 9: : Algorithm CI

Input: Users $\{U_1, \dots, U_n\}$ with their profiles $\{T_1, \dots, T_n\}$, and thresholds: δ

Output: C , communities of users

- 1: Construct a graph G by $\{T_1, \dots, T_n\}$ and δ ;
 - 2: **while** G is not empty **do**
 - 3: $T_i \leftarrow$ the highest degree node in G ;
 - 4: $L \leftarrow$ PSTs adjacent to T_i ;
 - 5: compute the node degree of the users in list L ;
 - 6: construct a community C_i which contains T_i ;
 - 7: **while** L is not empty **do**
 - 8: $T_j \leftarrow$ the highest degree node in L and remove T_j from L ;
 - 9: **if** T_j is adjacent to all the users in C_i **then**
 - 10: put T_j into group C_i ;
 - 11: **end if**
 - 12: **end while**
 - 13: insert group C_i into set C ;
 - 14: remove the PSTs in group C_i from graph G ;
 - 15: **end while**
-

PST	$N(T_i)$	Distance $\delta_{MST}(T_i, T_j)$	$ES(T_i)$
T_1	100	$\delta_{MST}(T_1, T_2) = 3.3$	7.1
		$\delta_{MST}(T_1, T_3) = 3.8$	
T_2	95	$\delta_{MST}(T_2, T_1) = 3.3$	6.4
		$\delta_{MST}(T_2, T_3) = 3.1$	
T_3	90	$\delta_{MST}(T_3, T_1) = 3.8$	6.9
		$\delta_{MST}(T_3, T_2) = 3.1$	

Table 5.1: Tree size and error sum of three PSTs

5.2.4 Selecting Representative PSTs

After identifying communities, one representative PST, denoted as r -PST, is selected for each community. Then, we can use r -PST to represent the profile for all profiles in a community.

To select an r -PST, there are two factors to be considered: one is the size of an r -PST and the other is the distance between the selected r -PST and other PSTs. A r -PST with the smaller size can not reduce the overhead of storing profiles for

PST	$\rho = 0.9$	$\rho = 0.5$	$\rho = 0.1$
T_1	0.350	0.349	0.348
T_2	0.331	0.323	0.315
T_3	0.318	0.327	0.335

Table 5.2: Example of selecting an r-PST.

a server but also provide better efficiency for query processing. For example, one can obtain the profile with different traveling behavior from him by accessing the profiles with larger distance values from the profile of his community. Thus, in this scenario, the smaller r-PSTs can speed up the computation for the distance values. Moreover, with a smaller distance between the r-PST and other PSTs in a group is, the more predication accuracy this r-PST could achieve. That is, this r-PST can best represent all other PSTs in the same community. Thus, a PST, which has a smaller tree size and is more similar to other PSTs in the same group, should be selected as r-PST.

For simplicity, the size of PST T_i is represented as the number of tree nodes, denoted as $N(T_i)$, and the error sum, denoted as ES , is used to quantify the distance between the r-PST and other PSTs in a group. Suppose that there are k PSTs in a group. The error sum of PST T_i is defined as the sum of the distance between T_i and other $k - 1$ PSTs. In other words, $ES(T_i)$ is formulated as $ES(T_i) = \sum_{j=1}^k \delta(T_i, T_j)$, where $\delta(T_i, T_i) = 0$. In order to take a balance between the tree size and the error sum, we can select the r-PST which can minimize

$$\alpha_i = \left\{ \rho \times \frac{N(T_i)}{\sum_{j=1}^k N(T_j)} + (1 - \rho) \times \frac{ES(T_i)}{\sum_{j=1}^k ES(T_j)} \right\},$$

where k is the number of PSTs in a community and $0 \leq \rho \leq 1$

Obviously, a parameter ρ can be used to give different weights to tree size and error sum. For example, consider a community with three users (i.e., U_1, U_2 and U_3) and their PSTs (i.e., T_1, T_2 and T_3). The tree size and the error sum of

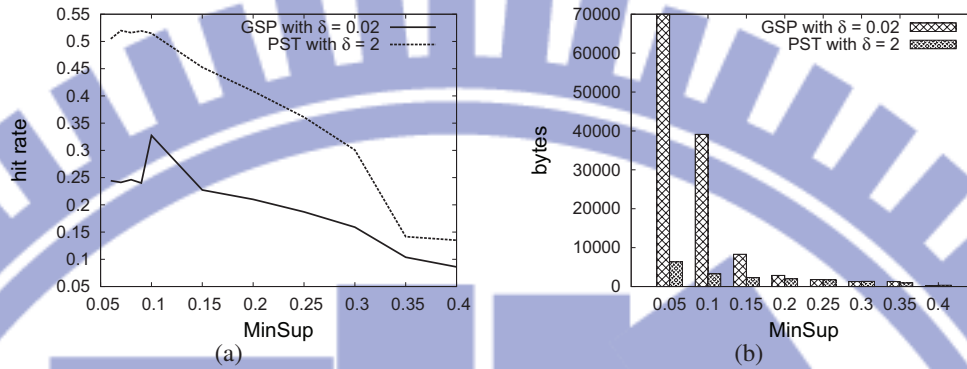


Figure 5.4: Comparison of GSP and our approach.

each PST is listed in Table 1. ρ is set to 0.9 and we can compute that $\alpha_1 = 0.9 \times \frac{100}{285} + (1 - 0.9) \times \frac{7.1}{20.4}$. By the similar fashion, we can obtain the results in Table 5.2 when ρ varied. It can be seen that T_2 is selected as the r-PST if we respect tree size and the error sum equally and set the parameter ρ to 0.5. In the case of $\rho = 0.9$, T_3 is selected as the r-PST because its tree size is the smallest among three PSTs. On the contrary, in the case of $\rho = 0.1$, T_2 is selected as the r-PST because of the least error sum among three PSTs.

5.2.5 Performance Comparison

In this section, we compare the hit rate and storage cost of GSP and our proposed approach PST. For fair comparison, we choose the best parameters for both approaches, where δ is set to be 2 and 0.02 for our approach PST and GSP, respectively.

Figure 5.4 shows the results with varied $MinSup$. The parameter $MinSup$ is used to distinguish the frequentness when deriving the sequential patterns and probabilistic suffix trees. In Figure 5.4(a), it can be seen that the hit rate of PST are higher than GSP in all cases. Interestingly, the hit rate of PST keeps constant when $MinSup$ is smaller than 0.1, while the hit rate of GSP has a peak when

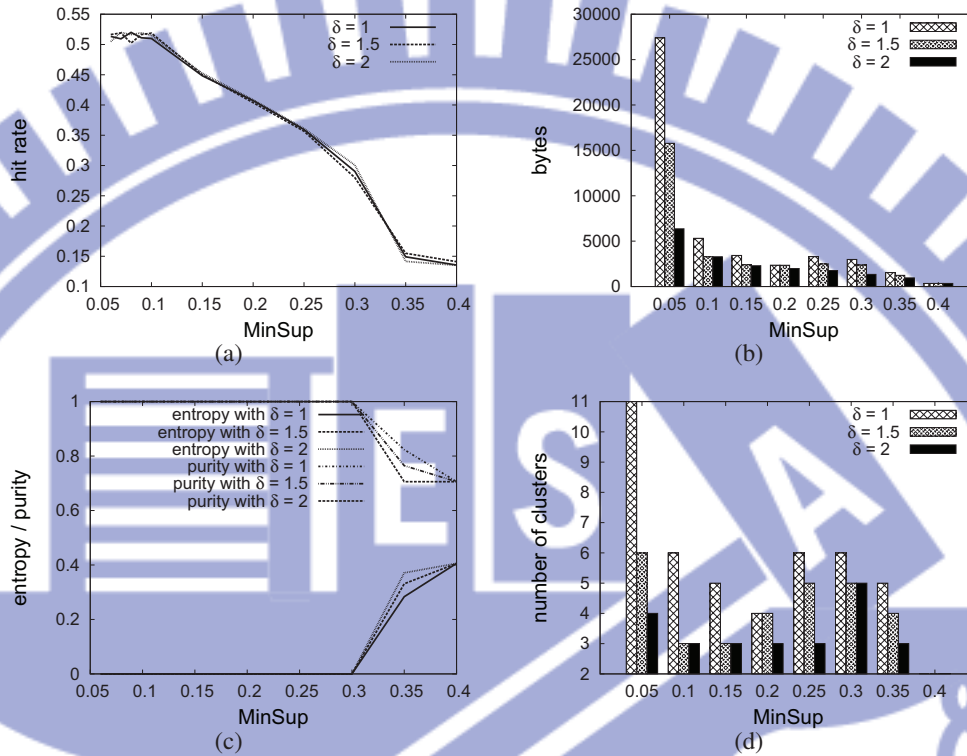


Figure 5.5: Sensitivity analysis when $MinSup$ varied.

$MinSup = 0.1$ and keeps decreasing when $MinSup$ increases. It is because that each source trajectory is used to generate 10 trajectories for a pilot such that the ground truth of $MinSup$ is almost 0.1. In Figure 5.4(b), it can be seen that the number of bytes of GSP are always larger than that of PST, especially in the case of $MinSup$ smaller than 0.15. From the experimental results, setting $MinSup = 0.1$ can bring the best performance of PST. Overall, from two experimental results above, PST outperforms GSP in terms of both storage and hit rate.


5.2.6 Sensitivity Analysis

In this section, we examine the impact of the threshold $MinSup$ of our approach. The sensitivity analysis of our approach should be discussed from four aspects.

Consider the cases when $MinSup$ is smaller than 0.15. Figure 5.5(a) shows that the hit rate is not sensitive to δ . Interestingly, Figure 5.4(b) shows that a smaller δ may lead to much larger storage. The threshold δ is used to identify community. Figure 5.5(d) shows that the number of clusters when $\delta = 1$ are more than other cases. However, from Figure 5.5(c), we know that entropy and purity are 0 and 1, respectively. It means that the users which have the similar profiles are divided into smaller clusters, thereby the hit rate being almost constant. On the other hand, consider the cases of $MinSup > 0.3$. Figure 5.5(a) shows that the hit rate decreases much significantly when $MinSup = 0.3$. Interestingly, Figure 5.5(d) shows that the number of clusters for $\delta = 2$ are correct (recall that the source trajectories are selected from three people), but Figure 5.5(c) shows that the quality of clustering decrease where the entropy and the purity increase and decrease, respectively. Thus, it leads to the significant decreasing of hit rate.

5.3 Conclusion

Nowadays, with the rapid development of positioning techniques, one can easily collect his trajectories by GPS. The interaction between people in these sites can be viewed as the location-based social networks where people can track and share location related information with each other. Mining community in such a novel social network can provide many interesting applications. For example, users can make friends with who have the same traveling interests, which can be achieved by finding people in the same community with him. On the other hand, the site can also recommend a user some interesting traveling pathes from the different communities. Consequently, in this paper, we target at the problem of mining community in a location-based social network. To achieve this goal, three main tasks should be done: 1. finding trajectory patterns for each user, 2. identifying

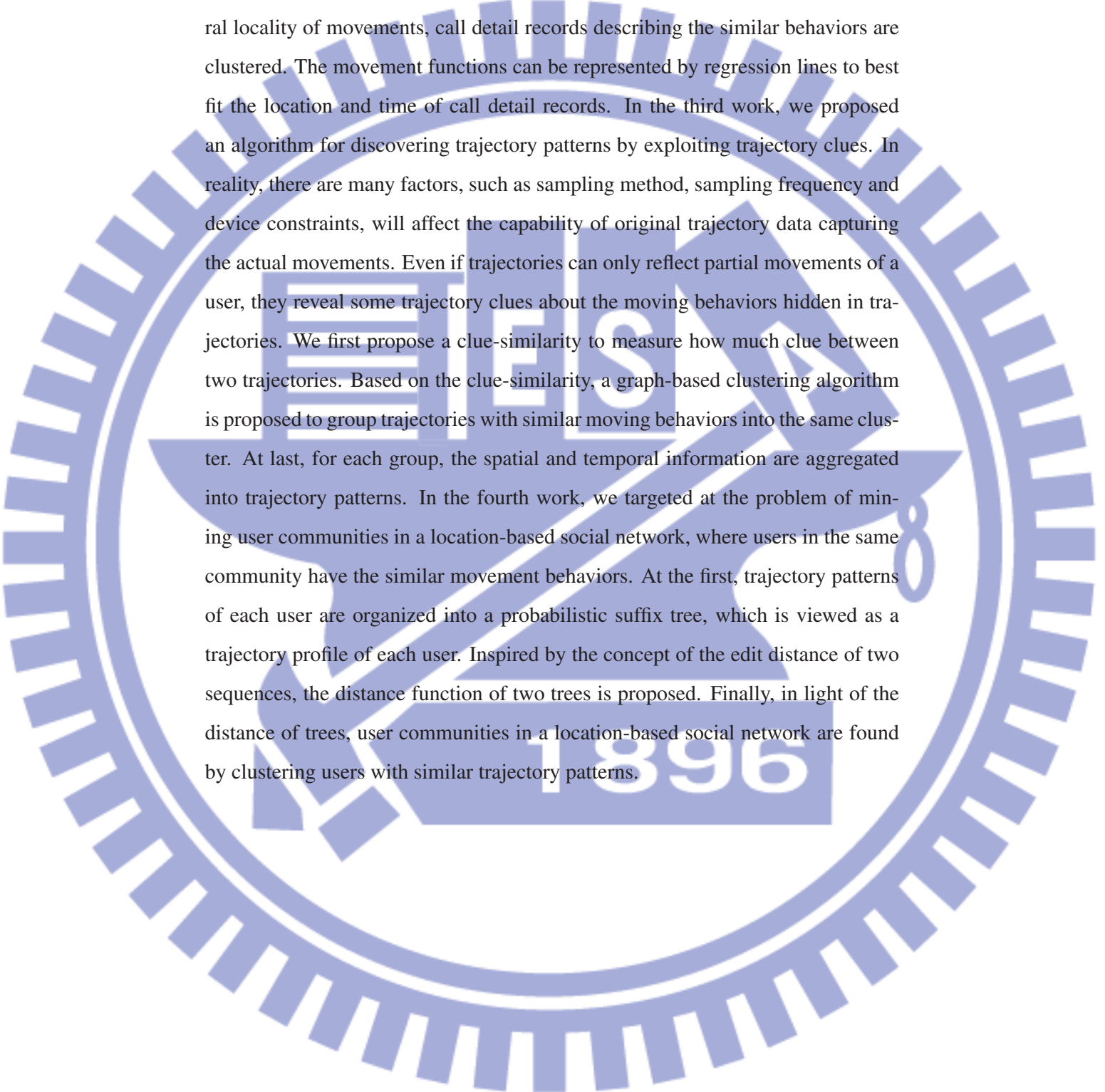
The background of the page features a large, light blue watermark of the Tsinghua University logo. The logo is circular with a gear-like outer edge. Inside the circle, there is a stylized figure holding a torch, and the year '1896' is prominently displayed in a white box at the bottom. The text of the paper is overlaid on this watermark.

the closeness of users by their patterns, and 3. clustering users with the similar trajectory patterns. However, prior works do not propose data structures to well-organize trajectory patterns, let alone clustering users by these trajectory patterns. Based on the observation, in this paper, we adopt a probabilistic suffix tree (abbreviated as PST) which can not only provide high accuracy for prediction the location of a user but also can build the profile of trajectory patterns on-the-fly. Specifically, a PST is used to generate trajectory patterns for each user. Since a PST can capture the moving behavior for a user precisely, we then derive the distance function for two PSTs to identify the community of users. For each community, we further select one representative PST to represent the profile of trajectory patterns in the same cluster. The representative PST can not only reduce the storage cost of maintaining many PSTs but also bring more benefit for applications, such as recommendation. Experimental results shows that our approach can not only effectively identify the community in a social network but also reduce the overhead for storing trajectory patterns efficiently.

Chapter 6

Conclusion

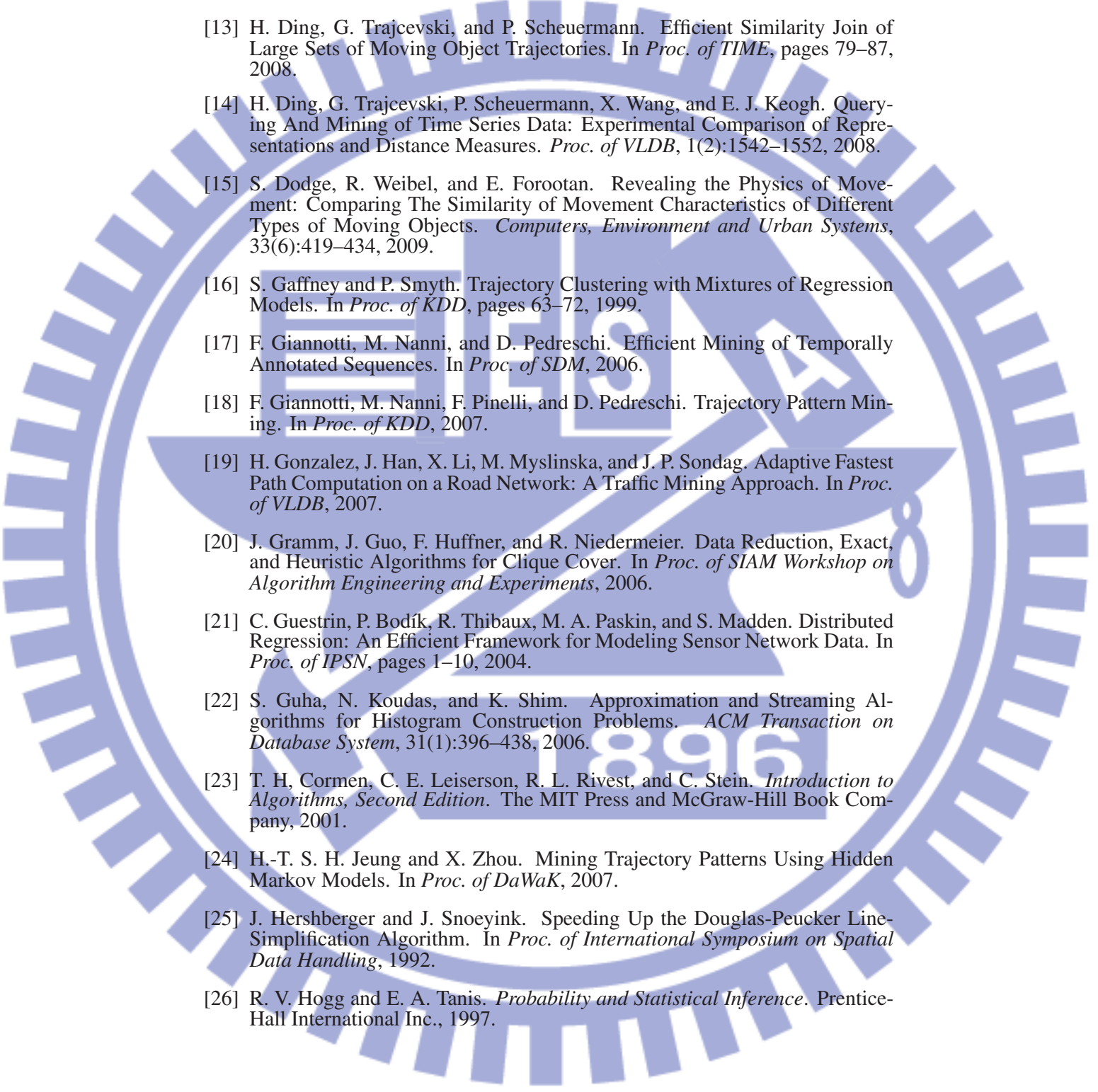
In this dissertation, we develop a series of research works for trajectory pattern mining and explore patterns mined for location-based social services. In our study, we present how to collect users' trajectories first. Then, two kinds of mining algorithms are proposed. Finally, we develop a framework for mining location-based social community structures. In the first work, we focused on the problem of data collection of trajectory data in a vehicular sensor network where every vehicles are equipped GPS and can communicate with each other in an ad-hoc manner. We proposed a framework MDC to reduce the amount of data transmission and the number of vehicles reporting their GPS data points. In MDC, model functions are derived to represent the raw GPS data points such that only some coefficients that describe its movements are reported. An in-network aggregation mechanism determines a set of groups and for each group, only one vehicle needs to report traffic data, thereby further reducing the number of simultaneous connections. In the second work, we proposed a regression-based approach to mine user movement patterns from call detail records in a mobile computing system. Call detail records are viewed as random sample trajectory data, and user movement patterns are represented as movement functions. At first, the call detail records that capture frequent user movement behaviors are extracted. By exploring the spatiotempo-





ral locality of movements, call detail records describing the similar behaviors are clustered. The movement functions can be represented by regression lines to best fit the location and time of call detail records. In the third work, we proposed an algorithm for discovering trajectory patterns by exploiting trajectory clues. In reality, there are many factors, such as sampling method, sampling frequency and device constraints, will affect the capability of original trajectory data capturing the actual movements. Even if trajectories can only reflect partial movements of a user, they reveal some trajectory clues about the moving behaviors hidden in trajectories. We first propose a clue-similarity to measure how much clue between two trajectories. Based on the clue-similarity, a graph-based clustering algorithm is proposed to group trajectories with similar moving behaviors into the same cluster. At last, for each group, the spatial and temporal information are aggregated into trajectory patterns. In the fourth work, we targeted at the problem of mining user communities in a location-based social network, where users in the same community have the similar movement behaviors. At the first, trajectory patterns of each user are organized into a probabilistic suffix tree, which is viewed as a trajectory profile of each user. Inspired by the concept of the edit distance of two sequences, the distance function of two trees is proposed. Finally, in light of the distance of trees, user communities in a location-based social network are found by clustering users with similar trajectory patterns.

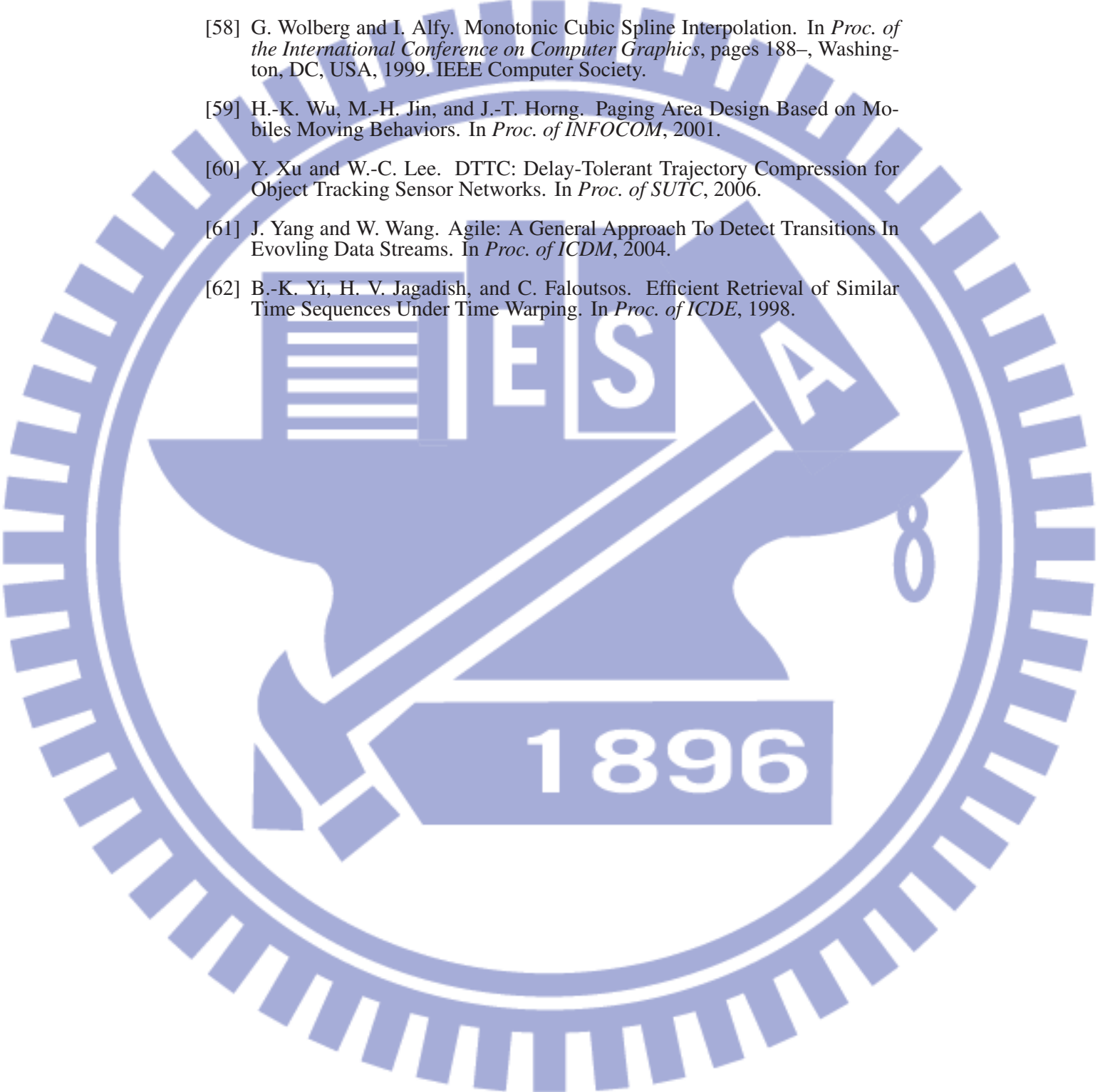
Bibliography

- [1] EveryTrail - GPS Travel Community. <http://www.everytrail.com/>.
- [2] MapMyRun Website. <http://www.mapmyrun.com>.
- [3] Run GPS Community Server. <http://www.gps-sport.net/>.
- [4] T. Brinkho. *Network-based Generator of Moving Objects*. Technical Report of the Institut für Angewandte Photogrammetrie und Geoinformatik (IAPG), [available] <http://www.fh-oow.de/institute/iapg/personen/brinkhoff/generator/>.
- [5] H. Cao, N. Mamoulis, and D. W. Cheung. Mining Frequent Spatiotemporal Sequential Patterns. In *Proc. of ICDM*, 2005.
- [6] H. Cao, O. Wolfson, and G. Trajcevski. Spatio-temporal Data Reduction with Deterministic Error Bounds. *VLDB Journal*, 15(3):211–228, 2006.
- [7] L. Chen and R. T. Ng. On The Marriage of Lp-norms and Edit Distance. In *Proc. of VLDB*, 2004.
- [8] L. Chen, M. T. Özsu, and V. Oria. Robust and Fast Similarity Search for Moving Object Trajectories. In *Proc. of SIGMOD*, 2005.
- [9] Y. Chen, M. A. Nascimento, B. C. Ooi, and A. K. H. Tung. SpADe: On Shape-based Pattern Detection in Streaming Time Series. In *Proc. of ICDE*, pages 786–795, 2007.
- [10] D. Chu, A. Deshpande, J. M. Hellerstein, and W. Hong. Approximate Data Collection in Sensor Networks Using Probabilistic Models. In *Proc. of ICDE*, pages 48–59, 2006.
- [11] D. J. Dailey, F. W. Cathey, and S. Pumrin. An Algorithm to Estimate Mean Traffic Speed Using Uncalibrated Cameras. *IEEE Transactions on Intelligent Transportation Systems*, 1(2):98–107, June 2000.
- [12] A. Deshpande, C. Guestrin, S. Madden, J. M. Hellerstein, and W. Hong. Model-Driven Data Acquisition in Sensor Networks. In *Proc. of VLDB*, pages 588–599, 2004.

- 
- [13] H. Ding, G. Trajcevski, and P. Scheuermann. Efficient Similarity Join of Large Sets of Moving Object Trajectories. In *Proc. of TIME*, pages 79–87, 2008.
- [14] H. Ding, G. Trajcevski, P. Scheuermann, X. Wang, and E. J. Keogh. Querying And Mining of Time Series Data: Experimental Comparison of Representations and Distance Measures. *Proc. of VLDB*, 1(2):1542–1552, 2008.
- [15] S. Dodge, R. Weibel, and E. Forootan. Revealing the Physics of Movement: Comparing The Similarity of Movement Characteristics of Different Types of Moving Objects. *Computers, Environment and Urban Systems*, 33(6):419–434, 2009.
- [16] S. Gaffney and P. Smyth. Trajectory Clustering with Mixtures of Regression Models. In *Proc. of KDD*, pages 63–72, 1999.
- [17] F. Giannotti, M. Nanni, and D. Pedreschi. Efficient Mining of Temporally Annotated Sequences. In *Proc. of SDM*, 2006.
- [18] F. Giannotti, M. Nanni, F. Pinelli, and D. Pedreschi. Trajectory Pattern Mining. In *Proc. of KDD*, 2007.
- [19] H. Gonzalez, J. Han, X. Li, M. Myslinska, and J. P. Sondag. Adaptive Fastest Path Computation on a Road Network: A Traffic Mining Approach. In *Proc. of VLDB*, 2007.
- [20] J. Gramm, J. Guo, F. Huffner, and R. Niedermeier. Data Reduction, Exact, and Heuristic Algorithms for Clique Cover. In *Proc. of SIAM Workshop on Algorithm Engineering and Experiments*, 2006.
- [21] C. Guestrin, P. Bodík, R. Thibaux, M. A. Paskin, and S. Madden. Distributed Regression: An Efficient Framework for Modeling Sensor Network Data. In *Proc. of IPSN*, pages 1–10, 2004.
- [22] S. Guha, N. Koudas, and K. Shim. Approximation and Streaming Algorithms for Histogram Construction Problems. *ACM Transaction on Database System*, 31(1):396–438, 2006.
- [23] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.
- [24] H.-T. S. H. Jeung and X. Zhou. Mining Trajectory Patterns Using Hidden Markov Models. In *Proc. of DaWaK*, 2007.
- [25] J. Hershberger and J. Snoeyink. Speeding Up the Douglas-Peucker Line-Simplification Algorithm. In *Proc. of International Symposium on Spatial Data Handling*, 1992.
- [26] R. V. Hogg and E. A. Tanis. *Probability and Statistical Inference*. Prentice-Hall International Inc., 1997.

- 
- [27] B. Hull, V. Bychkovsky, Y. Zhang, K. Chen, M. Goraczko, A. Miu, E. Shih, H. Balakrishnan, and S. Madden. CarTel: A Distributed Mobile Sensor Computing System. In *Proc. of SenSys*, pages 125–138, 2006.
- [28] C.-C. Hung and W.-C. Peng. Clustering Object Moving Patterns for Prediction-Based Object Tracking Sensor Networks. In *Proc. of CIKM*, 2009.
- [29] H. Jeung, Q. Liu, H.-T. Shen, and X. Zhou. A Hybrid Prediction Model for Moving Objects. In *Proc. of ICDE*, 2008.
- [30] H. Jeung, Q. Liu, H. T. Shen, and X. Zhou. A Hybrid Prediction Model for Moving Objects. In *Proc. of ICDE*, 2008.
- [31] H. Jeung, M. L. Yiu, X. Zhou, C. S. Jensen, and H. T. Shen. Discovery of Convoys in Trajectory Databases. *Proc. of VLDB*, 1(1):1068–1080, 2008.
- [32] M.-H. Jin, J.-T. Horng, M.-F. Tsai, and E. H.-K. Wu. Location Query Based on Moving Behaviors. *Information Systems*, 32(3), 2007.
- [33] P. Kalnis, N. Mamoulis, and S. Bakiras. On Discovering Moving Clusters in Spatiotemporal Data. In *Proc. of SSTD*, 2005.
- [34] E. Kanoulas, Y. Du, T. Xia, and D. Zhang. Finding Fastest Paths on a Road Network with Speed Patterns. In *Proc. of ICDE*, 2006.
- [35] E. J. Keogh. Exact Indexing of Dynamic Time Warping. In *Proc. of VLDB*, 2002.
- [36] K. M. Kockelman and J. Ma. Freeway Speeds and Speed Variations Preceding Crashes, Within and Across Lanes. *The Transportation Research Forum*, 46(1):42–61, Spring 2007.
- [37] D.-L. Lee, J. Xu, B. Zheng, and W.-C. Lee. Data Management in Location-dependent Information Services: Challenges and Issues. *IEEE Pervasive Computing*, 1(3), 2002.
- [38] J. G. Lee, J. Han, and K. Y. Whang. Trajectory Clustering: A Partition-And-Group Framework. In *Proc. of SIGMOD*, pages 593–604, 2007.
- [39] Z. Li, J.-G. Lee, X. Li, and J. Han. Incremental Clustering for Trajectories. In *Proc. of DASFAA*, pages 32–46, 2010.
- [40] Y.-B. Lin. Modeling Techniques for Large-scale PCS Networks. *IEEE Communications Magazine*, 35(2), 1997.
- [41] Y.-B. Lin and A.-C. Pang. *Wireless and Mobile All-IP Core Networks*. John Wiley, 2005.
- [42] C.-H. Lo, W.-C. Peng, C.-W. Chen, T.-Y. Lin, and C.-S. Lin. CarWeb: A Traffic Data Collection Platform. In *Proc. of MDM*, 2008.
- [43] C.-H. Lo, W.-C. Peng, C.-W. Chen, T.-Y. Lin, and C.-S. Lin. CarWeb: A Traffic Data Collection Platform. In *Proc. of MDM*, 2008.

- 
- [44] N. Mamoulis, H. Cao, G. Kollios, M. Hadjieleftheriou, Y. Tao, and D. W. Cheung. Mining, Indexing, and Querying Historical Spatiotemporal Data. In *Proc. of SIGKDD*, 2004.
- [45] M. Nanni and D. Pedreschi. Time-Focused Clustering of Trajectories of Moving Objects. *Journal of Intelligent Information Systems*, 27(3):267–289, 2006.
- [46] W.-C. Peng and M.-S. Chen. Developing Data Allocation Schemes by Incremental Mining of User Moving Patterns in a Mobile Computing System. *IEEE Transactions on Knowledge and Data Engineering*, 15(6), 2003.
- [47] W.-C. Peng and M.-S. Chen. Shared Data Allocation in a Mobile Computing System: Exploring Local and Global Optimization. *IEEE Transactions on Parallel and Distributed Systems*, 16(4), 2005.
- [48] W.-C. Peng, Y.-Z. Ko, and W.-C. Lee. On Mining Moving Patterns for Object Tracking Sensor Networks. In *Proc. of MDM*, 2006.
- [49] D. Ron, Y. Singer, and N. Tishby. The Power of Amnesia: Learning Probabilistic Automata with Variable Memory Length. *Machine Learning*, 25(2-3):117–149, 1996.
- [50] Saurabh Amin et al. *Mobile Century - Using GPS Mobile Phones as Traffic Sensors: A Field Experiment*. The 15th World Congress on Intelligent Transportation Systems, 2008.
- [51] S. Subramaniam, T. Palpanas, D. Papadopoulos, V. Kalogeraki, and D. Gunopulos. Online Outlier Detection in Sensor Data Using Non-Parametric Models. In *Proc. of VLDB*, pages 187–198, 2006.
- [52] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison Wesley, 1995.
- [53] W. L. Tan, F. Lam, and W. C. Lau. An empirical study on the capacity and performance of 3g networks. *IEEE Transaction on Mobile Computing*, 7(6):737–750, 2008.
- [54] G. Trajcevski, H. Ding, P. Scheuermann, R. Tamassia, and D. Vaccaro. Dynamics-aware Similarity of Moving Objects Trajectories. In *Proc. of GIS*, 2007.
- [55] S.-M. Tseng and C.-F. Tsui. An Efficient Method for Mining Associated Service Patterns in Mobile Web Environments. In *Proc. of SAC*, 2003.
- [56] F. Verhein and S. Chawla. Mining Spatio-temporal Association Rules, Sources, Sinks, Stationary Regions and Thoroughfares in Object Mobility Databases. In *Proc. of DASFAA*, 2006.
- [57] M. Vlachos, M. Hadjieleftheriou, D. Gunopulos, and E. J. Keogh. Indexing Multidimensional Time-Series. *VLDB Journal*, 15(1):1–20, 2006.

- 
- [58] G. Wolberg and I. Alfy. Monotonic Cubic Spline Interpolation. In *Proc. of the International Conference on Computer Graphics*, pages 188–, Washington, DC, USA, 1999. IEEE Computer Society.
- [59] H.-K. Wu, M.-H. Jin, and J.-T. Horng. Paging Area Design Based on Mobiles Moving Behaviors. In *Proc. of INFOCOM*, 2001.
- [60] Y. Xu and W.-C. Lee. DTTC: Delay-Tolerant Trajectory Compression for Object Tracking Sensor Networks. In *Proc. of SUTC*, 2006.
- [61] J. Yang and W. Wang. Agile: A General Approach To Detect Transitions In Evolving Data Streams. In *Proc. of ICDM*, 2004.
- [62] B.-K. Yi, H. V. Jagadish, and C. Faloutsos. Efficient Retrieval of Similar Time Sequences Under Time Warping. In *Proc. of ICDE*, 1998.