PATTERN MINING IN SPATIOTEMPORAL DATABASES

SHENG CHANG

(M.Eng. XIAN JIAOTONG UNIVERSITY, CHINA)

A THESIS SUBMITTED

FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2010

Acknowledgements

First of all, I gratefully acknowledge my supervisors, Professor Wynne Hsu and Professor Mong Li Lee. I thank them for their persistent support and continuous encouragement, for sharing with me their knowledge and experience. During the period of my Ph.D. study, they not only provided constant academic guidance and insightful suggestions to my research, but also taught me how to overcome difficulties with an optimistic attitude.

I wish to thank Dr. Joo Chuang Tong, Dr. See Kiong Ng, Dr. Xing Xie and Dr. Yu Zheng, whom I worked with on various research topics. I thank them for providing many fruitful discussions and valuable comments, as well as the datasets for the experiments in my research work. I also thank Professor Anthony K. H. Tung, Professor Sung Wing Kin and Professor Kian-Lee Tan. As my thesis advisory committee members, they provided constructive advice on my thesis work.

I would like to thank my parents for their efforts to provide me with the best possible eduction and their continuous moral support and encouragement during my long period of study. I hope I will make them proud of my achievement.

Last but not least, I would also like to thank the people in School of Computing for always being helpful over the years. I thank my friends at the National University of Singapore for their help.

Table of Contents

Summary

Advances in sensing and satellite technologies and the rapid spread of moving devices generate a large volume of spatiotemporal data of different types and promote the development of spatiotemporal database, thereby arising an increasing need for discovering spatiotemporal patterns in spatiotemporal data. To date, although a lot of works have been proposed for mining patterns in spatiotemporal databases, there are some research areas that need further investigation. In this thesis, we focus on efficiently and effectively discovering the spatiotemporal patterns in three popular spatiotemporal data types: biological sequence data, snapshot data and moving object data. We outline our approaches as follows.

First, we study the problem of mining mutation chains in biological sequences which are associated with location and time. We propose a mutation model where each biological sequence influences its spatiotemporal nearby biological sequences. We therefore define the notion of mutation chains and design an efficient algorithm to mine frequent mutation chains. Second, we tackle the problem of discovering localized and time-associated patterns in snapshot data. We propose an influence model where each object exerts an influence to its spatiotemporal nearby regions. Based on the influence model, we investigate this problems in two steps: We introduce the global Spatial Interaction Patterns (SIPs) on a single snapshot and propose a grid based influence model to mine the frequent SIPs. We further extend the SIPs to Geographical-specific Interaction Patterns (GIPs) and propose a quadtree based influence model and an efficient mining algorithm to mine frequent GIPs over time. Finally, we address the problem of discovering duration-aware trajectory patterns in moving object data for trajectory classification. The influences of moving objects to the regions are measured by the amount of time spent by the moving objects in the regions. Based on the influence, we introduce the duration-sensitive region rules and a top-down region partition approach to discover valid region rules. We also introduce the speed-differentiating path rule and propose a trajectory network to facilitate the mining of discriminative path rules. Two classifiers, TCF and TCRP, are built using the discovered region rules and path rules. Experiment results on real-world datasets show that both classifiers outperform the existing classifiers.

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, we witnessed the rapid development of sensing and satellite technologies and tracking devices, which significantly changed and are changing our world. The high spatial and spectral resolution remote sensing systems and other monitoring devices are gathering vast amounts of data with location and time attributes. These spatiotemporal data are stored and managed in spatiotemporal databases. This, in turn, leads to interest in spatiotemporal data mining.

Spatiotemporal data mining aims to disclose insightful knowledge embedded in spatiotemporal data, and enables people to understand the underlying process in spatiotemporal phenomena, and enables decision makers to make policies for emerging spatiotemporal events. To users, interesting spatiotemporal phenomena are those that are not random but rather follow certain rules. We call the repeating regular structures in space and time as spatiotemporal patterns.

Different types of spatiotemporal data have different regular structures, thereby having different spatiotemporal patterns. Spatiotemporal patterns are important because they not only are insightful knowledge but also can be applied for further data analysis and knowledge discovery. This thesis focuses on the spatiotemporal pattern mining in three popular types of spatiotemporal data.

1.1 Spatiotemporal Database

A spatiotemporal database deals with either geometry changes over time in discrete steps, or location of objects in a continuous manner [18]. Accordingly, the spatiotemporal data can be divided into moving object data and non-moving data.

The moving object data record the continuous location sequences of moving objects, where the location sequence of each object can be represented by a trajectory. The non-moving data record the information of spatial objects over time in discrete steps, where the spatial objects are distinct from each other. Further, the non-moving data can be modelled as events or snapshots. Event data record the discrete spatial objects over time. A point-based event is a spatial object which is tagged with the exact spatial and temporal information. The biological sequences which are associated with location and time can be treated as the point-based event data. Snapshot data record the distribution of spatial objects over time. Each snapshot is a time slice to record the distribution of spatial objects.

1.1.1 Biological sequence data

Biological sequence analysis is one of the major research area in the biomedical and bioinformatics. The biomedical applications generate a large volume of biological sequences. A biological sequence is a single, continuous molecule of nucleic acid or protein. Besides the biomolecular sequence (nucleic acid or protein), the annotation information (organism, species, function, spatiotemporal information, mutations linked to particular diseases, bibliographic, etc) are also stored in biological sequence databases [3].

Due to the annotated spatiotemporal information, each biological sequence can be seen as one point-based event in spatiotemporal space, which is associated with a sequence of molecules. Figure 1.1 shows an example of the biological sequence database, which consists of seven biological sequences, and its distribution in spatiotemporal space.

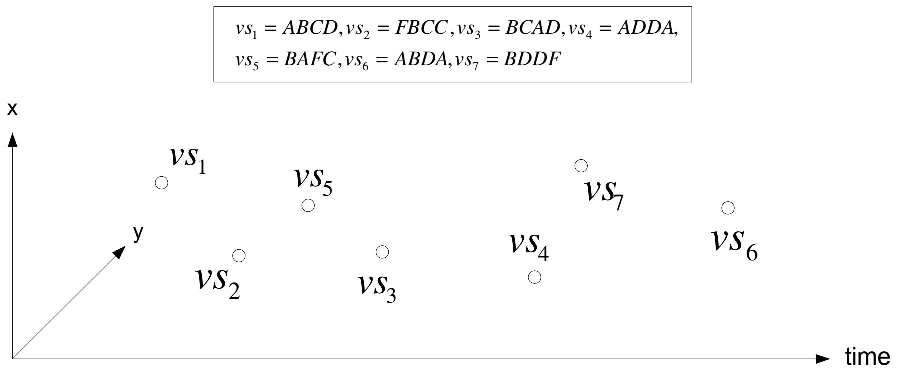

$vs_1 = ABCD, vs_2 = FBCC, vs_3 = BCAD, vs_4 = ADDA,$
$vs_5 = BAFC, vs_6 = ABDA, vs_7 = BDDF$
x
y
time
vs_1
vs_5
vs_7
vs_6
vs_4
vs_2
vs_3

Figure 1.1: Sequences data

1.1.2 Snapshot data

Studies of El Nino effects in meteorology, forest fires in forestry, volcanic activities and earthquake zones in geophysics, vegetation evolution in botany, generate a large volume of images that capture the spatiotemporal phenomena. For example, botanists maintain a historical record of the spatial distribution of trees to analyze the spatial patterns of vegetation [5]. Another major source of snapshot data is from web related applications. A web site may record a large volume of geographical information such as providers' locations, content locations, serving locations [76], and visitors' locations and visiting time.

These data are represented as a sequence of snapshots where each snapshot is associated with a spatial plane and a time slice, and contains a set of spatial objects. Figure 1.2 shows several snapshots over time, where each snapshot is a spatial plane during a time period.

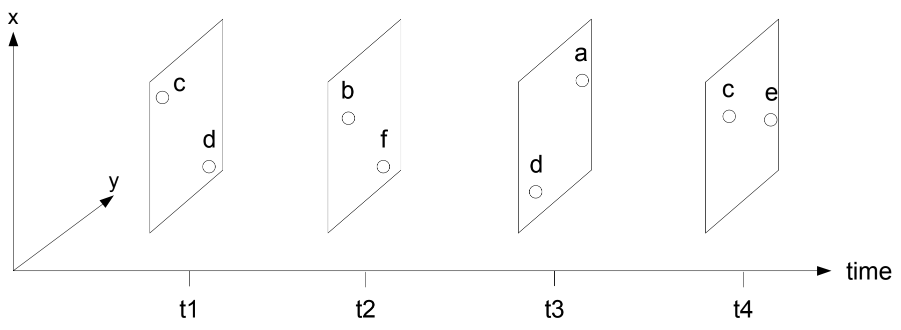


Figure 1.2: Snapshot data

1.1.3 Moving object data

In the applications which emphasize on the behavior of objects, moving object data are generated and managed in databases for online object tracking and future trajectory analysis. In meteorology, meteorologists maintain the data of moving storms, developments of high pressure areas and precipitation areas in the spatiotemporal database. In zoology, zoologists maintain animal movements, mating behavior, species relocation and extinction in the spatiotemporal database. In our daily living, traffic department and commercial companies store the trajectories of cars, trucks and taxis.

Moving object data are the time-ordered sequences of object locations. Figure 1.3 shows the geographical projection of tropical storm tracking trajectory data on the North Atlantic Ocean during 1950-2008, which are the linear segments of sampling points. Besides, moving object data may contain other affiliation information about the objects. Figure 1.3 shows the speeds and scales of tropical storms where blue trajectories are gentle tropical storms and red trajectories are hurricanes.

1.2 Motivations

While there have been some research works that focus on the pattern mining in biological sequence data, snapshot data and moving object data, more works need to be done.

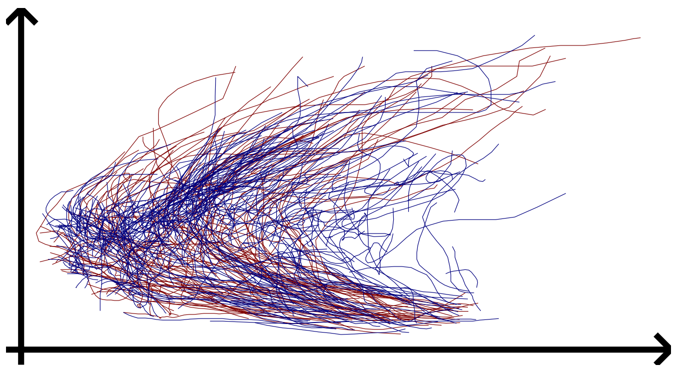


Figure 1.3: Moving object data

In this thesis, we explore the challenges of mining spatial patterns and spatiotemporal patterns in biological sequence data, snapshot data and moving object data, respectively.

1.2.1 Pattern mining in biological sequence data

To date, researches on sequence data mainly focus on the frequent patterns of sequences such as sequential pattern. Sequential patterns [2, 88, 54] are the frequent subsequences in a sequence database. Sequential pattern mining has received long-term research attention, because sequential patterns have broad applications including the analysis of long-term customer purchase behaviors for cross selling and target marketing, the analysis of Web access patterns for understanding user behaviors, the analysis of sequencing or time-related processes such as scientific experiments, natural disasters, and disease treatments, the analysis of patients' medical records, the analysis of biological sequences such as genome sequences and protein sequences, and so on.

There is no research which focuses on the spatiotemporal relationship of sequences. Taking spatiotemporal behaviors into account is important to better understand the biological sequence mutations. For example, influenza is a major human pathogen and the influenza virus, in existence for centuries, has been continually infecting both humans and animals. A recent trend is to develop region-specific vaccines which requires the spatial and temporal dynamics of the viral mutations. Thus, it is highly desirable to find out when and where the mutations occur, i.e., we need to know the highly-mutated regions (hotspots) in sequences at one geographical location and their changes when the sequence mutates in another location.

The spatiotemporal patterns of biological sequences are complex because they not only detect the highly-mutated regions in sequences but also identify the temporal chains of changes. Extending existing sequential pattern mining algorithms [2, 88, 54] or existing spatiotemporal event sequence mining algorithm [33] to find these complex spatiotemporal patterns is not feasible due to the large search space of highly-mutated regions in sequences and temporal dimensions. Therefore, it is desirable to formally define and efficiently mine the frequent spatiotemporal patterns of biological sequences.

1.2.2 Mining spatiotemporal patterns in snapshot data

Many applications, such as epidemiology and web services, have sustained interest in developing techniques to discover the localized patterns for performing further regional analysis and providing Location-Based-Services (LBS). The localized patterns may change over time, which leads to the chains of localized patterns.

For example, a comprehensive web site contains a large number of web pages, which are categorized into different topics such as news, sports, entertainment, and so on. The web site designer wants to know the visitors' interests in different countries/regions. If geographical-specific interests are discovered, the web site can pack the specific topic combinations for the visitors of specific countries/regions, and provide customized advertisements to different regions.

The traditional approaches to define the spatial relationship of events on snapshots are based on either the grid [72] or the Euclidean distance [31]. The grid based approach performs a preprocessing which imposes a grid on the spatial plane, transforms the events into transactions, and applies the well-developed transaction based pattern mining algorithms. The grid based approach is efficient, but it is inappropriate for spatial data due to the spatial information loss during preprocessing. On the other hand, the Euclidean distance based approach evaluates the spatial relationship by first computing the distance for every event pair and then counting the close pairs. A typical Euclidean distance based pattern is the spatial collocation pattern [63, 31] which is the set of event types whose events occur close together. In spite of no spatial information loss in Euclidean distance based approach, it is computationally expensive to compute the pairwise event distances. In addition, the discovered patterns are sensitive to the distance threshold and imprecise spatial data. Therefore, we need an interestingness measure which can identify the spatial relationship, handle imprecise data and do not rely on the grid.

The localized patterns (patterns with confined locations) and their changes over time are crucial to understand the spatiotemporal phenomena in snapshot data. However, there is no research work which focuses on such localized pattern mining. It is inappropriate to first mine the local patterns on the sub-datasets and then combine the local patterns, because it is difficult to determine the granularity of sub-datasets and is expensive to discover many intermediate patterns. We need an approach which does not rely on the existing geographical domain knowledge like hierachical region structures, and can automatically determine the region granularity. In addition, the localized patterns and their changes are complex because the patterns contain spatial and temporal information, which leads to a huge number of candidate patterns. We need the efficient algorithms to prune the candidate pattern space and discover the localized patterns and their changes.

1.2.3 Mining spatiotemporal patterns for trajectory classification

Trajectory classification is an important research problem in trajectory data analysis. Assume each trajectory in the trajectory database has a class label, trajectory classification is the process of predicting the class labels of moving objects based on their trajectories and other features.

The ability to classify trajectories is useful in many real world applications. In meteorology, a trajectory classifier can predict the intensity and scale of an approaching hurricane, so that precautionary actions can be carried out in advance. In homeland security, it is reported that more than 160,000 vessels are travelling in the United States' waters [45], and an anomaly trajectory detection classifier that can evaluate the vessels' behaviors and highlight suspicious vessels for further monitoring is highly desirable.

Existing work on trajectory classification [42] selects the regions and representative trajectories as the features for classification. Regions are mined based on the spatial distribution of trajectories, and representative trajectories are mined based on the shapes of trajectories. However, it does not take the duration of the trajectories into consideration in differentiating the objects that move at different speeds. For example, the speed at which a tropical hurricane passes the Gulf of Mexico is an important criterion in classifying the scale and intensity of the trajectories in Figure 1.3. Classifiers, that look only at the spatial distributions and movement directions of hurricanes but ignore the moving speeds, are unable to accurately classify the intensities of the hurricanes.

Spatiotemporal patterns which focus on both the actual movement paths and the movement speeds are desirable to build the trajectory classifier. However, few existing works considered the duration information in the moving object data analysis. Existing works on moving clustering [46, 36], motion group [81] and convoy [35] focus on the discovery of moving objects which exhibit synchronous movement behaviors. Even with low support, the paths of moving clusters or motion groups may not be enough for classification. This happens especially in the database where the moving objects are unlikely to move simultaneously, such as the annual hurricane trajectory database and the shuttle bus trajectories.

The trajectory patterns [25] are duration associated patterns which capture the Region-of-Interests (RoIs) and the transition time between every two RoIs. The mining of trajectory patterns is based on the pre-defined popular regions and transform the trajectories into region sequences. Having a pre-determined granularity for regions and duration intervals is undesirable because if the granularity is too coarse, it will lead to a small number of trajectory patterns which is not enough to build an accurate classifier. On the other hand, if the granularity is too fine, it will lead to a large number of trajectory patterns, resulting in overfitting. Hence, the trajectory patterns do not have discriminative power for accurate classification.

1.3 Contributions

This thesis is organized as follows. Figure 1.4 shows the overall framework. In this figure, the spatiotemporal data is further categorized into biological sequence data, snapshot data and moving object data. Figure 1.4 includes the spatial pattern mining layer, the spatiotemporal pattern mining layer, and the other data mining task layer to address the three data mining problems above.

The first problem is the mutation chains mining in biological sequence data based on a spatiotemporal constraint. The second problem is the discovery of localized and time-associated spatial relationships in snapshot data, where the spatial relationships are presented by interaction patterns. The third problem focuses on the discovery of the region rules and path rules in moving object data and the application of these rules for trajectory classification. The three major contributions are summarized as follows.

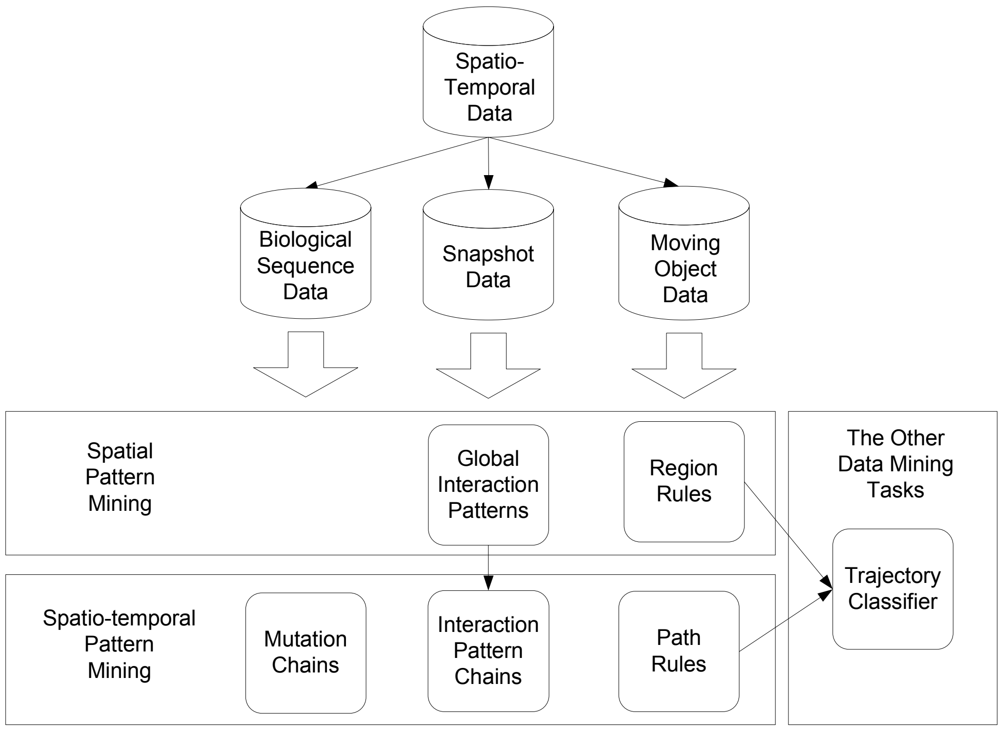


Figure 1.4: Thesis Framework

1. We propose a mutation model for biological sequence data where each biological sequence influences the other nearby biological sequences. Based on this mutation model, we define the problem of mining mutation chains and introduce a measure called mutation index to capture the confidence of a mutation. We present an integrated algorithm to discover contiguous subsequences of mutations. The algorithm utilizes two data structures to facilitate the mining process. The PointMutation tree summarizes position-specific single character mutations while the compact MaxMutation tree is designed to store the complete set of contiguous subsequences of mutations (k-mutations). We propose two pruning strategies to improve the mining efficiency. The first strategy prunes positions which are impossible to have any valid mutations based on the lower and upper bounds of their entropy measures. The second strategy is a selective join that enables us to prune unnecessary sequence chains based on the previous rounds of mining

results. We evaluate the algorithms on both synthetic and real world datasets. Experiments on the real world Influenza A virus database provide insights into the spread and mutation of the highly pathogenic Avian H5N1 influenza virus and the recent H1N1 swine flu. This work is published in [67].

2. We propose an influence model for snapshot data where each object exerts influence to its nearby regions. The influence model is able to capture the underlying spatial relationship among objects on the snapshot. Based on the influence model, we investigate the problem of discovering localized and time-associated patterns by two steps. First, we mine the global Spatial Interaction Patterns (SIPs) on single snapshot. We propose a grid based influence model and design an algorithm called PROBER to discover SIPs. We design the interaction tree structure to store the possible combination of candidate spatial interaction patterns, and extend PROBER algorithm to mining maximal SIPs. Second, we extend SIPs to the Geographical-specific Interaction Patterns (GIPs) over continent snapshots. We propose a quadtree based influence model and design an algorithm called FlexiPROBER to discover the localized GIPs. We define three pattern trends, i.e., enlargement, shrinkage and movement of supporting regions, to capture the changes in these patterns and develop an algorithm called MineGIC to discover these changes. Experiment results on both synthetic and real world datasets demonstrate that the proposed approach is effective in mining the local geographical-specific interests patterns and discover their changes over time. This work is published in [65, 64].

3. We propose duration-sensitive region rules and speed-differentiating path rules for trajectory classification. We propose that the influences of moving objects to the regions are measured as the time spent by the moving objects in the regions. Based on this influence definition, we propose a top-down region parti-

tion approach to discover the valid region rules. We also introduce the trajectory network to model the distribution of trajectory database. The granularity is controlled and measured by the Minimum Description Length (MDL) gain. Based on the trajectory network, we design a path pattern tree to enumerate the candidate path patterns, and design an efficient path pattern mining algorithm to mine the top-*k* covering path rules. Two classifiers, TCF and TCRP, are built using the discovered region rules and path rules. Experiment results on real-life trajectory datasets show that both TCF and TCRP obtain higher classification accuracy than the existing classifier. This work is submitted to conference for review [66].

1.4 Organization of the Thesis

The rest of this thesis is organized as follows. Chapter 2 reviews the related work on sequential pattern mining, pattern mining in snapshot data and spatiotemporal mining in moving object data. Chapter 3 proposes a mutation model and studies the mining of mutation chains in biological sequence database. Chapter 4 introduces the grid-based influence model and studies the mining of global interaction pattern in snapshot databases. Chapter 5 proposes a Quadtree based influence model and studies the mining of localized interaction patterns and further examines their enlargement, shrinkage and movement chains over space and time. Next, we consider the pattern mining in moving object data. Chapter 6 studies the discovery of duration-sensitive region rules and speed-differentiating path rules for trajectory classification. Two classifiers are built on those discovered rules. Finally, we conclude our studies and discuss some future work in Chapter 7.

Chapter 2

Related Work

Frequent pattern mining is an important research area in data mining. It focuses on discovering interesting knowledge in different data types, such as transactions, sequences, graphs, multimedia data and the other complex data types.

Agrawal et.al [1] first proposed to mine frequent item/itemset in transaction database and further discover association rules which are useful knowledge to discover the co-occurrence relationship among items. They applied the Apriori property to enumerate the candidate patterns and developed an efficient algorithm to mine all frequent patterns based on the Apriori property. As a paradigm in the area of data mining, the frequent pattern mining problem is explored and studied extensively.

Agrawal et.al [2] further proposed the sequential pattern mining problem. This problem is different from the association rule mining problem because sequential patterns are mined in sequence database, where each sequence is an *ordered* list of itemsets, instead of transactions in the association rule mining. Compared to the association rule mining problem, sequential pattern mining is more complex because the sequences contain more potential candidate patterns than transactions. A lot of work are proposed to efficiently find complete or compact set of sequential patterns, which will be surveyed in Section 2.1.

Spatiotemporal data are also *temporally ordered* sequences, but they contain more semantics than sequences due to the mixture of spatial and temporal information. Hence, spatiotemporal pattern mining is more complex and challenging than sequence pattern mining. First, the conventional frequent pattern mining approaches and algorithms need to be modified to perform efficient mining. Second, the discovered spatiotemporal patterns are expected to include spatial and temporal information.

However, existing work [1, 2, 72] on spatiotemporal mining are the direct extension of the conventional pattern mining in transactions or sequences. They usually employ a preprocessing step to transform spatiotemporal data into transactions or sequences, and then apply the existing pattern mining algorithms on the transformed data. This is undesirable because the transformed data may miss a lot of important spatial information during this preprocessing step. For example, two spatially close objects may fall into two different buckets using gridding spatial partition approach. In this chapter, we review the related work on sequential pattern mining, on pattern mining in event databases, and finally on data mining in moving object data.

2.1 Sequential pattern mining

Sequential pattern mining problem can be stated as "given a sequence database and the min support threshold, sequential pattern mining is to find the complete set of sequential patterns in the database" [29]. Some important definitions in this area are listed as follows. An itemset i is denoted by $(i_1 i_2 \ldots i_m)$, where i_j is an item. A sequence s is denoted by $\langle s_1 s_2 \ldots s_n \rangle$ where s_j is an itemset. A sequence $\langle a_1 a_2 \ldots a_n \rangle$ is contained by another sequence $\langle b_1 b_2 \ldots b_m \rangle$ if there exist integers $i_1 < i_2 < \ldots < i_n$ such that $a_1 \subseteq b_{i_1}, a_2 \subseteq b_{i_2}, \ldots, a_n \subseteq b_{i_n}$. For example, the sequence $\langle (bd)(c)(ac) \rangle$ is contained in $\langle (e)(bd)(ae)(c)(b)(acd) \rangle$, since $(bd) \subseteq (bd), (c) \subseteq (c), (ac) \subseteq (acd)$. Table 2.1 gives an example of sequence database which contains four transactions.

TID	sequences
10	$\langle(e)(bd)(ae)(c)(b)(acd)\rangle$
20	$\langle(bd)(c)(ac)\rangle$
30	$\langle(d)(ac)(abc)\rangle$
40	$\langle(e)(g)(af)(c)(b)(c)\rangle$

Table 2.1: An example of sequence database

From Table 2.1, we can see that each sequence is temporally ordered itemsets and the sequential patterns are the frequent subsequences in the sequence database. Similar to frequent patterns, sequential patterns have the anti-monotone (i.e., downward closure) property as follows: every non-empty sub-sequence of a sequential pattern is a sequential pattern. In other words, if a sequence S is infrequent, none of the super-sequences of S will be frequent. For example, suppose $\langle hb\rangle$ is infrequent, all of its super-sequences, such as $\langle hab\rangle$ or $\langle h(bc)\rangle$, are infrequent. Based on this anti-monotone property, the sequential pattern mining focuses on the development of efficient algorithms to discover the sequential patterns.

GSP [71] is a sequential pattern mining algorithm based on a horizontal data format. It adopts a multiple-pass, candidate-generation-and-test approach in sequential pattern mining. The first database scan determines the support of each item, and every frequent item yields a 1-element frequent sequence. After the initialization of 1-item sequences, GSP utilizes the sequential pattern of k-item to generate new potential patterns of (k+1)-item, called candidate sequences. GSP carries out one database scan to collect support count for candidate sequences. All candidates whose support in the database are no less than minimal support form the set of the newly found sequential patterns. The algorithm terminates when no new sequential pattern is found in a pass, or no candidate sequence can be generated. However, GSP still generates a large number of candidates and requires costly multiple database scans.

SPADE [88] is an Apriori-Based vertical data format sequential pattern mining algorithm. SPADE maps a sequence database into the vertical data format which takes each item as the center of observation and takes its associated sequence and event identifiers as data sets. Similar to GSP, SPADE generates the (k+1)-length candidate patterns by joining two frequent k-length sequential patterns. The SPADE algorithm reduces the access of sequence databases since the information required to construct longer sequences are localized to the related items and/or subsequences represented by their associated sequences and event identifiers. However, the basic search methodology of SPADE is similar to GSP, exploring both breadth-first search and Apriori pruning.

PrefixSpan [53] is a write-based sequential mining algorithm. PrefixSpan uses frequent items to recursively project sequence databases into a set of smaller projected databases and grow subsequence fragments in each projected database. To reduce the length of projected sequences, PrefixSpan examines only the prefix subsequences and project only their corresponding postfix subsequences into projected databases. PrefixSpan counts the supports of candidate patterns in the projected database. The mining algorithm terminates when no new projected database is generated or no new sequential pattern is found. PrefixSpan is reported to outperform GSP and SPADE because the projected databases are much smaller than the whole database.

The sequential pattern mining methodology has also been extended to handle different application scenarios. To handle incremental mining problem, IncSpan [10] defines an intermediate state between frequent patterns and infrequent patterns called semi-frequent patterns. Given min_sup, and a factor $\mu \leq 1$, a sequential pattern is semi-frequent if its support falls in the range $[\mu * min\_sup, min\_sup)$. With the incremental updating of sequence database, the patterns may transform among different states, from infrequent to semi-frequent, from semi-frequent to frequent, etc. Based on the state transformation, IncSpan proposes some pruning strategies to prune the search space of sequential patterns. To handle the noisy environment where the items of sequence data may be imprecise, [84] has studied the problem of mining frequent sequences with the help of the compatibility matrix, which provides a probabilistic connection from the observation to the underlying true value.

All the related work above need to generate a complete set of candidate patterns during the mining. The performance of such algorithms often degrades dramatically when mining long frequent sequences, or when using very low support thresholds. To tackle this problem, CloSpan [82] is proposed to mine frequent closed sequential patterns, i.e., those containing no super-sequence with the same support, instead of mining the complete set of frequent subsequences. CloSpan performs an early termination on the prefix search tree when finding the backward sub-patterns or super-patterns. However, setting min support is a subtle task in the sequential pattern mining algorithms. TSP [73] is proposed to discover top-k closed sequences. TSP finds the most frequent patterns early in the mining process and allows dynamic raising of minimal support which is then used to prune unpromising branches in the search space.

Many researchers [23, 55, 56] shift their attention towards mining sequences by incorporating constraints to reduce search space. [23] proposes regular expressions as constraints for sequence pattern mining and develops a family of SPIRIT algorithms while members in the family achieve various degrees of constraint enforcement. Following that, [55, 56] conducts a systematic study on pushing various constraints deep into sequential pattern mining and characterizes constraints for sequential pattern mining according to their application semantics and roles in sequence pattern mining

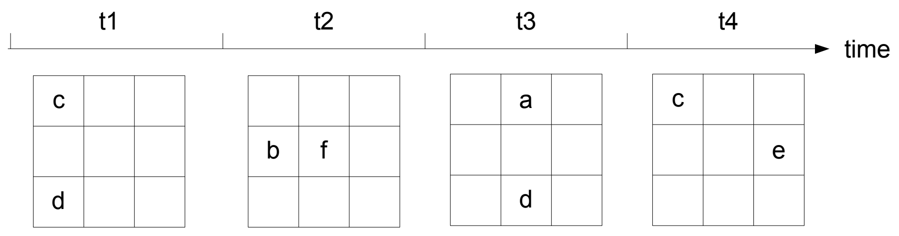
Sequential pattern mining, which focuses on the temporal relationship of itemsets, has been studied extensively. However, existing work on sequential pattern mining do not consider the spatial relationship and spatial information in the mining. It is infeasible to transform the spatiotemporal data into sequence data by mapping the regions into items of sequences. This is because the mapping mechanism results in inevitable information loss in spatiotemporal data. Hence, sequential pattern mining is unable to be directly applied to mine spatiotemporal patterns. The pattern mining in spatiotemporal data is more complicated than sequential pattern mining due to the mixture of temporal and spatial relationship.

2.2 Pattern mining in event data

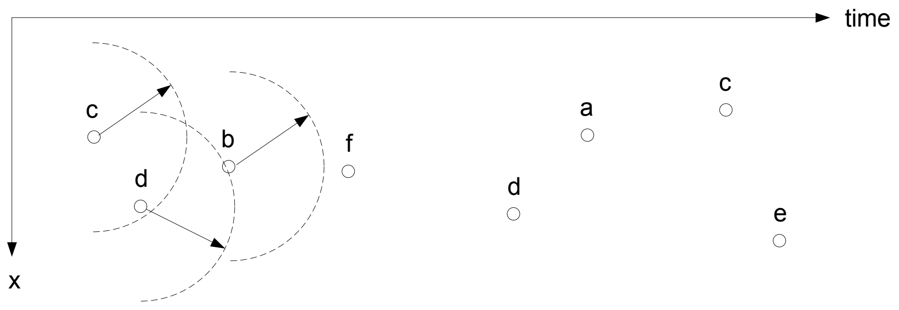
Spatiotemporal event data are a collection of events in the space-time dimensions, where each event is associated with a set or a sequence of event type. Typically, spatiotemporal event data come from GIS, meteorology applications and web logs. Spatiotemporal patterns mining in event data will discover the frequent patterns by measuring spatiotemporal relationship among events. There are many research work in this research area. Depending on the methods to measure spatiotemporal relationship among events, the existing work can be classified into two categories.

- Snapshot-grid. Snapshot-grid model assigns a spatial snapshot for each time slice along the time axis and links all spatial planes together with chronological order. Snapshot-grid model imposes a grid on each spatial snapshot, relying on a domain knowledge or cell granularity if no domain knowledge. Figure 2.1(a) shows an example of spatiotemporal data which are stored and accessed by snapshot-grid model. Based on the snapshot and grid, the spatiotemporal data are easily transformed into transactions of cell ids, so that the conventional pattern mining techniques [1, 2] can be seamlessly employed in spatiotemporal data mining.

- Event model. Event model emphasizes the mutual relation of event pairs and a global relation of dataset through some existing distance functions and similarity measure. The relation does not rely on any domain knowledge but a spatiotemporal distance definition. Figure 2.1(b) shows an example of event model, in which the events are in X-dimension and time-dimension for easy illustration and dash arcs are boundaries of the event influence range. The spatial access techniques are usually employed to facilitate the access and computation of spatiotemporal distance.



(a) Snapshot-grid model



(b) Event model

Figure 2.1: An example of spatiotemporal database

2.2.1 Snapshot-grid model

[72] is an early work of spatiotemporal mining based on grid-snapshot model. In this work, Tsoukatos et.al impose a grid on the spatial plane so that each event is represented by a cell id of the grid. A spatiotemporal sequential pattern has the form $IS_1 \rightarrow IS_2 \rightarrow \ldots \rightarrow IS_n$ to describe a frequent event sequence, where two neighboring items have both spatial and temporal constraints. More specifically, $IS_1, \ldots, IS_n$ all occur in the same cell and each neighboring item pair, IS_{i-1} and IS_i, happens in two consecutive snapshots. This work utilizes a lattice structure to enumerate candidate sequential patterns. Tsoukatos et.al proposes the algorithm DFS_MINE to mine

all maximal sequential patterns in the depth first search manner. The limitations of this work are that the patterns largely depend on the pre-imposed grid and all items in a sequential pattern must occur in the same cell, which is a strong spatial constraint.

Flow patterns [79, 78] partially alleviate the spatial constraint in the spatiotemporal sequential patterns [72]. Like spatiotemporal sequential patterns, a flow pattern also has the form $IS_1 \rightarrow IS_2 \rightarrow \ldots \rightarrow IS_n$. Each neighboring item pair, IS_{i-1} and IS_i, occurs in two neighboring cells (or the same cell) and happens in two consecutive snapshots, which is a relaxed spatiotemporal constraint. Compared to spatiotemporal sequential patterns, flow patterns contain more knowledge due to the relaxed spatiotemporal constrain, which also lead to great increase of candidate patterns. An Apriori-like algorithm FlowMiner is proposed to efficient mine the flow patterns. However, FlowMiner still relies on the pre-defined spatial neighbor definition based on grid.

The pervious two works partition the spatial plane by imposing a uniform grid, Verhein et.al [74] further alleviate this limitation by allowing the use of domain knowledge to manually partition the spatial plane. They define the spatio-temporal regions, stationary regions and high traffic regions, and further define the spatio-temporal association rules called STAR, denoted by $(r_i, TI_i, q) \rightarrow (r_j, TI_{i+1})$, where r_i and r_j are dense regions, TI_i and TI_{i+1} are two consecutive time intervals and q is a selection predicate. The algorithm STAR-miner mines spatio-temporal association rules by devising a pruning property based on the high traffic regions. However, in spite of the flexibility of non-uniform region partition, their work falls into the category of grid model, which, as mentioned, may cause information loss.

In summary, grid model is a simple but effective model to transform spatial data into spatial identifier, such that the spatiotemporal mining may be simplified and some existing pattern mining algorithms could be applied directly. However, grid model has two major problems, which are summarized as follows. First, grid model is not adaptive to handle different datasets. It needs different pre-knowledge or cell granularity to handle different datasets. Second, grid model is not robust to handle uncertain data, while the uncertainty is ubiquitous in spatial data because of both equipment limitations and man-made error.

2.2.2 Event model

Spatial association rule [41] is an early work on the research of spatial relationship among spatial event. The work defines a set of predicates such as “adjacent_to”, “close_to”, “within” and so on, and a set of hierarchies for data relations. This rule can describe how frequent one or more predicates occur in the spatial database. Based on the hierarchical topology relations, the proposed solution converts spatial database into transaction database, such that spatial association rules mining problem is transformed into conventional association rule mining.

Spatial collocation pattern [63, 51, 31] is another kind of spatial patterns. This pattern presents a set of event types which are frequently located close to each other, and its statistical foundation is based on Ripley’s K function [58, 14]. Spatial collocation pattern is first proposed in [63] and further improved in [31]. Their solutions are based on the *event centric model*, where an *instance* of a pattern P is a set of objects that satisfy the unary (feature) and binary (neighborhood) constraints specified by the pattern’s graph. For example, $\{a_1, b_1, c_1\}$ is an instance supporting the clique pattern $P = \{a, b, c\}$, if the distance of any two instances is not more than the given threshold σ. Two measures, participation ratio and prevalence, are developed to evaluate the candidate patterns. The prevalence is a monotonic measure to allow iterative pruning. Based on these concepts, an Apriori-like approach called Co-location Miner is developed to find all the frequent collocation patterns. Co-location Miner initially performs a spatial join to retrieve object pairs which are close to each other, and then it uses the Apriori-based candidate generation algorithm to generate the candidates of length (k+1)-pattern from k-patterns and validate the candidates by joining the instances of the k-patterns which share the first k-1 feature instances. Similarly, [51] studies the same problem to find sets of services located close to each other. This work also presents an Apriori-like algorithm. Different from Co-location Miner, it uses a Voronoi diagram and a quaternary tree to improve running time. However, the algorithms based on event centric model require an expensive spatial-join operation, so they are not scalable to the database size, i.e., the event number.

To alleviate the problem of event centric model, a few works [90, 87, 85, 77] focus on the issue of decreasing the number of spatial-join operation. Zhang et.al [90] utilize a space partitioning approach to partition the map into many buckets and distribute the events into corresponding buckets based on the event positions. The main advantage of this space partitioning approach is that one bucket maintains all possible neighbors for each event which is in this bucket. Hence, the mining algorithm can perform an independent spatial-join operation on each bucket and summarize the results of all buckets. Yoo et.al [87, 85] propose partial-join and join-less approaches to reduce the number of spatial-join operation. The key idea of both algorithms is to enumerate and sort all the neighboring instances into a projected database during the preprocessing phase, and focus on the projected database to prune instances. Similarly, Wang et.al [77] also employ the projected database to prune instances, but they propose a summary structure to store the necessary position information of events, and two hash-based indices to facilitate information retrieval operations in the summary-structure.

Celik et.al [8] extend the concept of collocation pattern and propose the mixed-drove spatiotemporal co-occurrence patterns which present the collocation patterns frequently over time. They employ a time prevalence to measure the time confidence of collocation pattern. They design a Apriori-like algorithm to prune the candidates. Yoo et.al [86] propose a co-evolving collocation patter query. Given a sequence of prevalence as a query, this work searches the collocation patterns whose normalized Euclidean distance between patterns' prevalence over the time and the query is less than a distance threshold. They employ lower bounding distance, instance-level upper bound and event-level upper bound to prune the candidate collocation patterns.

Huang et.al [32, 33] propose an extension of event model from spatial domain to spatiotemporal domain. Depending on the neighborhood parameters, each event has the spatiotemporal relationship to both spatially and temporally close events. They introduce the spatiotemporal sequential patterns of event data and use a sequence index as the significance measure for spatiotemporal sequential patterns. They propose an algorithm called Slicing-STSMiner for mining spatiotemporal sequential patterns. Slicing-STSMiner employs the temporal slicing to partition the data set into overlapping time slices, processes each slice separately, and recovers the whole patterns across slice boundaries due to the unidirectional property of time.

In summary, event model has less preprocessing than snapshot-grid model, but event model requires expensive spatial-join operations in mining, which is the major disadvantage of event model. Using the distance threshold as the spatiotemporal constraint, event model is also sensitive to noise data. This happens especially when the event distances are around the boundary of distance threshold. In addition, existing work on event model focus on the single feature events, not the combined feature events. Therefore, they can only discover long, single point sequences, i.e., sequences which occur multiple times at a specific position. They are unable to find sequential patterns which involve multiple features. In Chapter 3, we propose a novel event model based method to mine sequential patterns in event sequences.

2.3 Spatiotemporal mining in moving object database

The data mining of moving object data has emerged as a hot topic due to the increasing use of wireless communication devices. There are two sampling schemes to record the trajectories of moving objects: Uniform sampling and non-uniform sampling. Uniform sampling scheme records the object locations at every fixed time duration, while non-uniform sampling scheme records the object locations only if the velocity, the direction, or the other statuses change. Non-uniform sampling scheme greatly decreases the amount of data, but it results in greater research challenges because the snapshot model does not work on the non-uniform sampling data.

We also notice that the discovered patterns can be categorized into synchronous patterns and non-synchronous patterns. The synchronous patterns focus on the synchronous movement of some moving objects. The non-synchronous patterns focus on the common movement paths of moving object where they may not move together.

Based on the sampling scheme in moving object data and the presence of synchronousness in patterns, existing works can be classified into five categories as follows.

- Shape-based. The trajectory data do not include temporal information, so the data analysis and mining are performed on the shape of trajectories.
- Fixed duration and synchronous patterns (FD_SYN). The trajectories are sampled with the fixed time duration, and the data analysis and mining are performed based on snapshot analysis.
- Non-fixed duration and synchronous patterns (NFD_SYN): The trajectories are sampled with the non-fixed sampling rate so that the time durations between two sampling points may not be the same, and the data analysis and mining are performed based on a variant of snapshot analysis.

- Fixed duration and non-synchronous patterns (FD_NONSYN). The trajectories are sampled by a fixed time duration, and the data analysis and mining are performed in a non-synchronous event/object analysis manner.

- Non-Fixed duration and non-synchronous patterns (NFD_NONSYN). The trajectories are sampled by a non-fixed sampling rate, and the data analysis and mining are performed in a non-synchronous event/object analysis manner.

We summarize the related work of moving object data mining by the five categories above and four data mining tasks in Table 2.2.

Table 2.2: A summary of related work on moving object database mining

	Frequent Pattern mining	Clustering	Prediction	Classification
Shape-based	[7]	[22, 43]		[42]
FD_SYN	[48, 83]	[46, 36]	[34]	
NFD_SYN	[60, 35, 81]			
FD_NONSYN				
NFD_NONSYN	[25]		[50]	our work in Chapter 6

2.3.1 Frequent Trajectory Pattern Mining

In the category of shape-based, Cao et.al [7] study the problem of discovering the frequent repeated moving object paths based on the trajectory shapes. They do not utilize the grid partition strategy. Instead, they initially approximate trajectories by line segments, then discover frequent singular patterns from the segment set, finally perform mining using a substring tree. The output patterns are sequences ids, which are obtained from the influential regions of segments.

By snapshot based pattern mining, there are several existing work on mining sequential patterns in moving object databases. Mamoulis et.al [48] define periodic patterns in a long trajectory. Their solution is similar to snapshot-grid model. Given a grid, it partitions the entire spatial space S into n non-overlapping regions r_i, $1 \leq i \leq n$, such that $S = r_1 \cup r_2 \cup \ldots \cup r_n$ and for any two regions $r_i \cap r_j = \phi$, $1 \leq i, j \leq n$. The spatiotemporal data are translated into cell sequences. To overcome the disadvantage of snapshot-grid model mentioned in Section 2.2, Mamoulis et.al apply a density-based clustering to discover the dense clusters as the valid regions. To find spatiotemporal periodic patterns, they develop a two-phase top-down method. First, it uses a hash-based method to retrieve all frequent 1-patterns (i.e., a set of valid clusters), and replaces the trajectories in the database using cluster ids. Next, it uses the same methodology of maxsubpattern-tree algorithm to discover all the frequent patterns. Yang et.al [83] address the imprecise trajectories of moving objects since the sampling points are imprecise in real world applications. They apply the snapshot-grid model where the cell centers serve as the candidate locations of patterns, and propose a probability model to describe the uncertain support of pattern.

Two existing work focus on pattern mining in non-uniform sampling data. Sacharidis et.al [60] investigate the problem of maintaining hot motion paths, i.e., routes frequently followed by multiple objects over the recent past. Jeung et.al [35] focus on the discovery of object groups that have travelled together from some consecutive time intervals. They adopt a trajectory simplification technique to select the necessary snapshots for analysis. They apply the filter-and-refinement paradigm to reduce the overall computational cost. Similar to the convoy in [35], Wang et.al [81] introduce the valid group which is a group of moving users that are within a distance threshold from one another for at least a minimum time duration, but [81] focuses on the mining of maximal valid groups. An efficient algorithm called VGBK is proposed to identify maximal valid group by enumerating all maximal cliques in an undirected graph.

To discover non-synchronous movement in moving object data of non-fixed sampling rate, Giannotti et.al [25] introduce the trajectory patterns which are the sequence of dense areas associated with durations. During preprocessing, the dense areas named Region-of-interests (RoIs) are extracted and each trajectory is translated into a sequence of RoIs which are associated with the durations between two neighboring RoIs. Each trajectory is a temporally annotated sequences, so the frequent RoI sequences (i.e. trajectory patterns) are mined by the temporal-annotated pattern mining algorithm [24], which follows the projected database based sequential pattern mining paradigm. However, the main problem is how to select the proper parameters to control the granularity of Region-of-Interests (RoI), as too large granularity damage the pattern semantics and too small granularity results in a small number (or none in worst case) of patterns.

2.3.2 Trajectory Clustering

The early work of trajectory clustering is [22], in which Gaffney et.al propose a mixed model to cluster trajectories by considering a trajectory as a whole. Gaffney et.al utilize a probability density function to model the observed trajectories and adopt an Expectation-Maximization algorithm to train and obtain the local optimal probability density function. Lee et.al [43] propose a different approach which considers a partial trajectory i.e., segments, as the basic units for clustering. They propose a partition and group clustering framework which first partitions the trajectories into line segments guided by MDL principle, then groups the line segments using a variant of density based clustering algorithm. Both works above are based on the shapes of trajectories and do not consider the temporal information of moving object data.

Moving cluster detection [46, 36] considers the temporal information into clustering. Moving cluster is a group of objects in which a majority of members move together for some continuous snapshots. The main idea for this problem is to identify the clusters on snapshots by applying the existing clustering algorithms, like micro-clustering [46] and DBSCAN [36], and summarize the clusters which have common objects over time slices (snapshots). In [46], the bounding rectangles are employed to measure the compactness of the moving micro-clusters. If the size of the bounding rectangle exceeds a certain threshold, the micro-cluster is split. In contrast, Kalnis et.al [36] measure the similarity of two clusters by the percentage of common object identifiers. They not only propose two exact moving cluster detection algorithms, but also propose an approximate algorithm using grid and a process similar to video compression.

2.3.3 Moving Object Prediction

Moving object location prediction [34, 50] is seen as the application of moving object patterns based on one of the two assumptions that the moving object movement obeys its historical paths or the frequent paths of the other moving objects. Jeung et.al [34] predict the location based on the first assumption. They employ the periodic patterns [48] from the historic trajectory of the moving object to predict the positions of this object. A trajectory pattern tree is built to accelerate the pattern search in the later phase of prediction. Monreale et.al [50] predict the future position of moving objects based on the second assumption, i.e., the moving objects follow the common paths of the other moving objects. Monreale et.al employ trajectory patterns [25], which are the natural way to present such common paths of moving objects, to predict the future positions of moving objects. The trajectory patterns are organized in a compact structure called T-pattern tree to facilitate the prediction.

2.3.4 Trajectory Classification

Trajectory classification is a rather new research problem. Previous work on classifying trajectories are based on the feature vector (e.g., the maximum velocity, direction) derived from the whole trajectory [6]. The classification accuracy drops when handling complex trajectory datasets consisting of trajectories of varying lengths. To alleviate this problem, Lee et.al [42] propose a classification framework to partition the trajectories into line segments and derive two kinds of features, regions and representative trajectories. Regions are used to distinguish the activity areas of different trajectories, which are obtained by partitioning the spatial plane guided by MDL. The representative trajectories are used to summarize the common paths of different trajectories, which are obtained by the average paths of trajectory clusters. A major limitation of [42] is that both regions and representative trajectories are merely shape based features, which are not able to distinguish the moving objects of different velocities. In addition, the representative trajectories are skewed to the dominant trajectory class. In Chapter 6, we propose a novel classification approach based on the temporal information associated patterns.

Chapter 3

Mining Mutation Chains in Biological Sequences

Pattern mining in biological sequences helps in understanding the structure, function, and organization of cellular systems. Existing works on general sequential pattern mining [71, 88, 53] and biological sequential pattern mining [80, 30] are proposed to find the sequential patterns which are repeating subsequences in biological sequences. However, to the best of our knowledge, there is no existing work which considers the spatiotemporal relationship of biological sequences. This relationship is important to understand the virus mutation of infectious disease such as influenza.

The annual and occasionally pandemic influenza has become an alarming source of morbidity, mortality, and economic burden to the world. The influenza virus, in existence for centuries, has been continually infecting both humans and animals. It is able to do so because the genes of the influenza virus can change its protein coat (i.e., antigens) from time to time by mutation so as to find new susceptible non-immune populations to infect. In addition, the virus has an air-borne disease transmission mechanism which enables it to spread across geographical regions quickly. In the case of influenza A H5N1 virus, its natural reservoir in aquatic birds enables it to be spread rapidly over large distances geographically. The rapid increase in population density, air travel and interconnections between countries and continents further escalates the speed of disease transmissions.

Thus, it is highly desirable to present how mutations happen and when and where the mutations occur, i.e., we need to know the highly-mutated regions (hotspots) in virus sequences at one geographical location and their changes when the virus moves to another location. However, existing sequence mutation analysis [37] and phylogenetic analysis [16] cannot reveal the spread of mutation patterns as they do not correlate the mutations with where and when specific mutations have occurred. Existing sequential pattern mining algorithms [2, 88, 54] did not consider the spatiotemporal conditions.

In this chapter, we propose to mine mutation patterns which take the spatiotemporal features of sequences into account. The likelihood of whether a viral sequence mutates to another viral sequence is dependant on whether they occur within some time window period, the connectivity between the locations where they occur and their sequence similarity. We formally define the concept of *k*-mutation chains to present the two dimensional mutation patterns. We propose an efficient algorithm to mine *k*-mutation chains in biological sequences. We also apply the mining algorithm in the Influenza A virus database to discover mutation patterns.

3.1 Motivation

Mutations in influenza virus isolates have been found to be responsible for new outbreaks in Russia [40] and India [49]. Existing mutation analysis methods use sequence alignment and sequence comparison to identify point mutations [37]. They derive a model in the form of an amino acid translation probability matrix to estimate the future composition of amino acids. However, this model cannot reveal a virus's spread patterns as it does not correlate them with where and when specific mutations have occurred.

The fast-changing, fast-spreading virus render vaccines developed in advance of a pandemic to become less effective over time. A recent trend is to develop region-specific vaccines which requires the spatial and temporal dynamics of the viral mutations, i.e., we need to know how highly-mutated regions (hotspots) in virus sequences at one geographical location change when the virus moves to another location over time. We introduce the notion of **mutation chains** to capture mutation patterns with geographical spread over time.

Furthermore, current research has focused solely on identifying single-point mutations in the viral sequence. Single-point mutations are small mutations that alters only one nucleotide at a time. They are responsible for the so-called "antigenic drift", in which the virus gradually accumulates more and more such mutations, eventually causing them to become new strains. In contrast, "antigenic shift" is caused by a large and sudden mutation that involved the changing of many nucleotides, often leading to major outbreaks. This is because the more a virus has mutated, the more likely that the population's immune systems would not recognize it, and therefore would not have immunity to it. As such, in this chapter, we not only mine for single-point mutations (1-mutation), but we will also detect larger genetic changes that involved more than one (say, k) consecutive nucleotides (k-mutations) in the viruses' sequences as they spread over time and space.

Table 3.1: An example of virus protein sequence databases

ID	Year	Aligned Sequences	Country	Host
vs_1	1985	MNPNQKABCD	Mexico	Human
vs_2	1987	MNPNQKFBCC	USA	Human
vs_3	1988	MNPNQKBCAD	Canada	Human
vs_4	1989	MNPNQKADDA	Russia	Swine
vs_5	1988	MNPNQKBAFC	Spain	Human
vs_6	1993	MNPNQKABDA	Vietnam	Avian
vs_7	1991	MNPNQKBDDF	Iceland	Avian

Table 3.1 shows an example virus sequence database where each sequence is aligned with the others and a representative sequence segment of ten positions are shown for illustration. Positions 1 to 6 are conserved regions for the viruses because no mutation occurs at these six positions, while positions 7 to 10 are highly-mutated regions (hotspots). To further understand how sequences mutate at hotspots, we observe that vs_1 and vs_2 occur within a viable period of two years with a strong sequence similarity (i.e., eight positions have the same amino acids and two positions are different). These two virus are found in Mexico and USA, countries which share a common border. These factors provide evidence that vs_1 could possibly mutate to vs_2: “A” mutates to “F” at position 7, and “D” mutates to “C” at position 10. Extending this one step further, we observe that vs_2 has a relatively high sequence similarity with vs_5 (only positions 7, 8 and 9 are different) and although Spain is not geographically close to USA, there are extensive air travel patterns between the two countries. Since vs_5 occurs in Spain after vs_2 in USA, we suspect that the mutation spread could originate from Mexico and spread to USA, and further on to Spain. We denote this mutation chain as $\langle 7, \text{ABCD} \rightarrow \text{FBCC} \rightarrow \text{BAFC} \rangle$, where 7 indicates the start position of the hotspot.

A mutation chain is frequent if it is supported by an adequate number of sequence chains. Detecting frequent mutation chains is important to understand mutation behaviors which are vital to vaccine research targeting on the mutations. However, frequent mutation chain mining is computationally challenging because many sequences could possibly mutate to the others and the mutations could occur numerous times, leading to a large number of mutated sequence chains.

It is not feasible to use existing sequential pattern mining algorithms [2, 88, 54] to find mutation chains due to two reasons. First, sequential pattern mining is performed on the transactions, which are expensive to generate mutation chains as transactions to feed into sequential pattern mining algorithms. Second, sequential pattern mining is not position-specific, and after performing sequential pattern mining algorithms for each position in the virus sequence, we still require some way to efficiently combine the sequential patterns from different positions into the *k*-mutation chains. The existing spatiotemporal mining techniques [79, 33] can only discover long, single point mutations (i.e., mutations which occur multiple times at a specific position). They are unable to find co-mutations, which involve multiple positions. Combining the long, single point mutations again requires an expensive post-processing step.

In this chapter, we design a framework that integrates both horizontal (across multiple positions) and vertical mining (increasing depth of mutation chains) to perform early pruning of infeasible mutation chains, leading to algorithms that find highly mutated regions and identify how sequences mutate in virus sequences in a real world scenario. We take advantage of the growing availability of spatial and temporal information in public biological databases such as the SWISS-PROT protein sequence data bank [3] to discover meaningful mutation chains in the viral genetic sequence data. To the best of our knowledge, the problem of discovering patterns of mutations in the various genetic subtypes of the virus that takes into account of the spatial and temporal variations has not been explored by current bioinformatics research. We summarize the contributions of this work as follows:

- We define the problem of mining mutation chains and introduce a measure called mutation index to capture the confidence of a mutation.
- We present an integrated algorithm to discover contiguous subsequences of mutations. The algorithm utilizes two data structures to facilitate the mining process. The PointMutation tree summarizes position-specific single character mutations while the compact MaxMutation tree is designed to store the complete set of contiguous subsequences of mutations (k-mutations).
- We propose two pruning strategies to improve the mining efficiency. The first

strategy prunes positions which are impossible to have any valid mutations based on the lower and upper bounds of their entropy measures. The second strategy is a selective join that enables us to prune unnecessary sequence chains based on the previous rounds of mining results.

- We evaluate the algorithms on both synthetic and real world datasets. Experiments on the real world Influenza A virus database provide insights into the spread and mutation of the highly pathogenic Avian H5N1 influenza virus and the recent H1N1 swine flu.

The remainder of this chapter is organized as follows. Section 6.2 gives the preliminaries and problem statement. Section 3.3 presents both the bottom-up and the top-down mutation mining frameworks, and proposes two pruning techniques. In Section 3.4 we evaluate our algorithms on both synthetic datasets and real-world datasets. Finally, we conclude our work in Section 3.5.

3.2 Definitions and Problem Statement

A biological sequence database contains *n* tuples where each tuple comprises of the viral sequence *vs*, sequence id *sid*, as well as the location (x, y) and time *t* where the sequence was isolated or reported. The viral sequences are preprocessed by a multiple sequence alignment so that all sequences have identical number of positions where each position is an amino acid or a gap, denoted as "−". For protein sequences, 20 standard amino acids are available. Given a viral sequence, the amino acid or gap at *p*-th position is said to be the **p-th character** in the viral sequence. After alignment, the *p*-th character of a viral sequence will have a corresponding *p*-th character in another viral sequence.

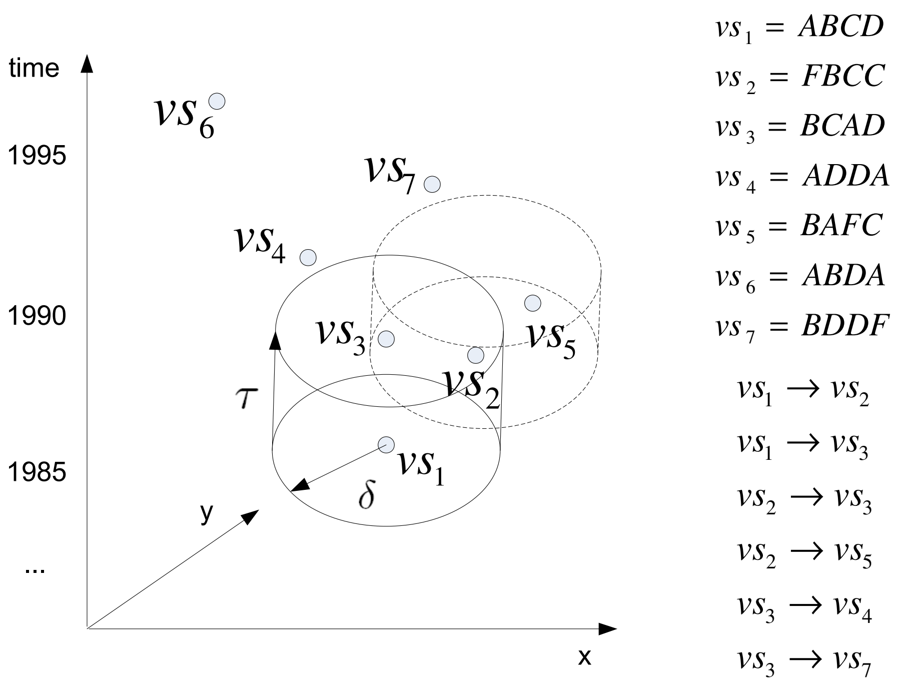


Figure 3.1: Example to show the likelihood of a virus mutating to another

We describe the mutation from a viral sequence to another viral sequence by position-specific character mutations. For example, if a sequence $A{-}CXE$ mutates to another sequence $ABCDE$, we say that A, C, E remain unchanged at positions 1, 3, 5 respectively, while B is inserted at position 2, and X is changed to D at position 4.

The likelihood of whether a viral sequence vs mutates to another viral sequence vs' is dependant on whether they occur within some time window period, the connectivity between the locations where they occur and their sequence similarity. Figure 3.1 shows the viral sequences from Table 3.1 with highly-mutated regions only, and a simplified likelihood model in the form of a cylinder with a circular base centered at (x, y), radius δ and height τ, denoting that a virus has a high probability of mutating to another virus if they occur in the same cylinder. From the two cylinders in Figure 3.1, vs_1 is likely to mutate to vs_2 and vs_3, while vs_2 is likely to mutate to vs_3 and vs_5. These mutations are indicated by $vs_i \rightarrow vs_j$.

We use $NB(vs)$ to denote the set of viruses that are likely to be mutated from vs and define point mutation as follows.

Definition 1. *Let c_p and c'_p be the p-th characters of sequences vs and vs′ respectively. c_p is said to **point mutate** or **1-mutate** to c'_p, if and only if $vs' \in NB(vs)$ and $c_p \neq c'_p$. We denote the point mutation at position p as $\langle p, c_p \rightarrow c'_p \rangle$.*

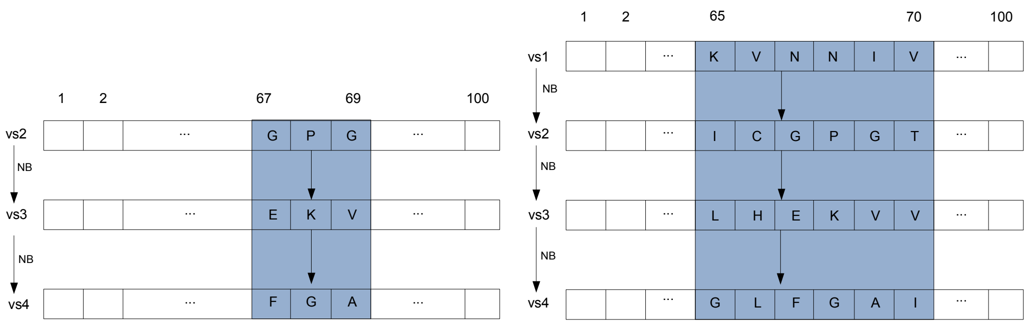
In addition to the point-mutations, longer mutations can also occur over a set of consecutive positions on the viral sequence. We introduce the notion of k-mutations to capture these:

Definition 2. *Let $c_p \ldots c_{p+k-1}$ be a subsequence of characters of viral sequence vs in position range [p, p+k-1], and $c'_p \ldots c'_{p+k-1}$ to be a subsequence of characters of viral sequence vs′ in position range [p, p+k-1]. $c_p \ldots c_{p+k-1}$ is said to **k-mutate** to $c'_p \ldots c'_{p+k-1}$, if and only if $vs' \in NB(vs)$ and $c_p \rightarrow c'_p$, $c_{p+1} \rightarrow c'_{p+1}$, ..., $c_{p+k-1} \rightarrow c'_{p+k-1}$. We denote this k-mutation starting at position p as $\langle p, c_p c_{p+1} \ldots c_{p+k-1} \rightarrow c'_p c'_{p+1} \ldots c'_{p+k-1} \rangle$.*

A sequence pair (vs_1, vs_2) is said to **support** the k-mutation $\langle p, c_p \ldots c_{p+k-1} \rightarrow c'_p \ldots c'_{p+k-1} \rangle$ if $vs_2 \in NB(vs_1)$ and the sequence of characters starting at position p in vs_1 and vs_2 corresponds to $c_p \ldots c_{p+k-1}$ and $c'_p \ldots c'_{p+k-1}$ respectively. For example, the sequence pair $(ABCD, CDMA)$ supports the 3-mutations: $\langle 1, ABC \rightarrow CDM \rangle$, $\langle 2, BCD \rightarrow DMA \rangle$, as well as the 4-mutations: $\langle 1, ABCD \rightarrow CDMA \rangle$.

As the mutation of the virus is an ongoing operation against the population's immunity system, a mutation could occur over multiple time points leading to a mutation chain. We define a k-mutation chain over T time points as follows.

Definition 3. *A **k-mutation chain** is given by $\langle p,\ s_1 \rightarrow s_2 \rightarrow \ldots s_i \rightarrow s_{i+1} \ldots \rightarrow s_T \rangle$ where s_i is said to be the **i-th state** in the k-mutation chain where s_i is a sequences of length k, $i \in [1, T]$. In a k-mutation chain $\langle p,\ s_1 \rightarrow s_2 \rightarrow \ldots s_i \rightarrow s_{i+1} \ldots \rightarrow s_T \rangle$, $s_i \rightarrow s_{i+1}$ denotes the i-th k-mutation $\langle p, s_i \rightarrow s_{i+1} \rangle$ where s_i and s_{i+1} are biological sequences of length k, $i \in [1, T-1]$.*



(a) A 3-mutation chain (b) A 6-mutation chain

Figure 3.2: Examples of k-mutation chains. The mutation chain in (a) is a sub-mutation of the mutation chain in (b)

A chain of sequences, $vs_1 \rightarrow vs_2 \ldots \rightarrow vs_T$, is said to **support** the k-mutation chain $\langle p, s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_T \rangle$, if (vs_i, vs_{i+1}) supports the i-th k-mutation $\langle p, s_i \rightarrow s_{i+1} \rangle$ for $\forall i \in [1, T-1]$. A k-mutation chain, $M_1 = \langle p, s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_T \rangle$, is a **sub-mutation** of a k′-mutation chain, $M_2 = \langle p', s'_1 \rightarrow s'_2 \rightarrow \ldots \rightarrow s'_{T'} \rangle$, denoted as $M_1 \sqsubseteq M_2$, if and only if

1. $p' \leq p \leq p' + k' - k$; $k \leq k'$; $T \leq T'$;

2. $\exists r \in [0, T' - T]$ and $\exists c \in [0, k' - k]$ such that $s_i(q) = s'_{i+r}(q + c)$ for $\forall i \in [1, T]$ and $\forall q \in [1, k]$, where $s_i(q)$ and $s'_{i+r}(q + c)$ are the q-th character of s_i and $(q + c)$-th character of s'_{i+r}, respectively.

Specifically, $M_1 = M_2$ if $M_1 \sqsubseteq M_2$ and $M_2 \sqsubseteq M_1$. For example, $\langle 1, AB \rightarrow BD \rangle \sqsubseteq \langle 0, CAB \rightarrow ABD \rangle \sqsubseteq \langle 0, ATGP \rightarrow CABC \rightarrow ABDT \rangle$.

Figure 3.2 shows a 3-mutation chain and a 6-mutation chain, and the 3-mutation chain is a sub-mutation of the 6-mutation chain. The mutations, indicated by the shaded regions, may be biologically significant since they are considered as hotspots and deserve further investigations.

Suppose we have a sequence chain (**CAB**ABDE → **ABD**BDFE) that supports the 3-mutation $\langle 1, CAB \rightarrow ABD \rangle$. AB and BD are substrings of CAB and ABD at position 2 respectively. Clearly, if a sequence supports $CAB \rightarrow ABD$ at position 1, it must support

$AB \rightarrow BD$ at position 2. Hence, if a sequence chain support a k-mutation M, it will support all M's sub-mutations.

Let $Support(M, i)$ be the collection of sequences belonging to a sequence chain supporting M and are on the i-th state of their respective chain. The **support set** of a k-mutation chain $M = \langle p, s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_T \rangle$, denoted as $SupportSet(M)$, is an ordered list of T sequence collections, $SupportSet(M) = [Support(M, 1), \ldots, Support(M, T)]$.

Definition 4. *The support of $M = \langle p, s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_T \rangle$, denoted as $Support(M)$, is defined as $Support(M) = min_{i \in [1,T]} |Support(M, i)|$.*

Figure 3.1 gives a BSD = {ABCD, FBCC, BCAD, ADDA, BAFC, ABDA, BDDF}. Let $M = \langle 2, BC \rightarrow CA \rightarrow DD \rangle$ to be a 2-mutation chain. Since there are two sequence chains in BSD: (A**BC**D, B**CA**D, A**DD**A) and (F**BC**C, B**CA**D, B**DD**F) that support M, we have $SupportSet(M) = [Support(M, 1), Support(M, 2), Support(M, 3)]$, such that $Support(M, 1)$ = {ABCD, FBCC}, $Support(M, 2)$ = {BCAD}, $Support(M, 3)$ = {ADDA, BDDF}. Then $Support(M) = \min\{2, 1, 2\} = 1$.

Definition 5. *Let BSD be a biological sequence database, and $M = \langle p, s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_T \rangle$ be a k-mutation chain. Let $Count(c, p)$ be the number of sequences in BSD that have character c at position p. We define the mutation ratio of string s_i $mRatio(M, i)$ as:*

$$mRatio(M, i) = \frac{|Support(M, i))|}{MAX_{q \in [1,k]}(Count(s_i(q), p + q - 1))}$$

where $s_i(q)$ is the q-th character of s_i.

The *mRatio* measures the fraction of sequences that supports mutation M at the i-th state to the sequence that happen to have the character $s_i(q)$ at position $p + q - 1$, where $1 \leq q \leq k$. Intuitively, a high value of *mRatio* at the i-th state indicates that the probability of the mutation occurring at the i-th state is high. Consider our running example $M = \langle 2, BC \rightarrow CA \rightarrow DD \rangle$. We compute the counts of characters B and C at positions 2 and 3 respectively:

$Count(B, 2) = |\{ABCD, FBCC, ABDA\}| = 3$

$Count(C, 3) = |\{ABCD, FBCC\}| = 2.$

With this, we compute the *mRatio* as follows:

$$\begin{aligned} mRatio(M, 1) &= \frac{|Support(M, 1)|}{MAX\{Count(B, 2), Count(C, 3)\}} \\ &= 0.67. \end{aligned}$$

Similarly, $mRatio(M, 2)$=1.0 and $mRatio(M, 3)$=0.67.

Definition 6. *The mutation index of* $M = \langle p, s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_T \rangle$*, denoted as* $mIndex(M)$*, is defined as* $mIndex(M) = \ min_{i \in [1,T]} \ mRatio(M, i)$

In our example, $mIndex(M) = min\{0.67, 1.0, 0.67\} = 0.67$. The mutation index is essentially a variant of all_confidence [52], a correlation measure satisfying anti-monotone property. Hence, the mutation index also satisfies anti-monotone property.

Lemma 1. ***Anti-monotonicity Property.*** *Given two mutation chains* $M_1 \sqsubseteq M_2$*,* $mIndex(M_2) \leq mIndex(M_1)$*.*

Proof: Let $M_1 = \langle p, s_1 {\rightarrow} s_2 {\rightarrow} \ldots {\rightarrow} s_T \rangle$ be a k-mutation chain, and $M_2 = \langle p', s'_1 {\rightarrow} s'_2 {\rightarrow} \ldots s'_{T'} \rangle$ be a k'-mutation chain. Without loss of generality, let $M_1 \sqsubseteq M_2$, so that we have 1) $\Delta p = p - p'$; 2) $\exists r$, $s_i(q) = s'_{i+r}(q + \Delta p)$ for $1 \leq q \leq k$, $1 \leq i \leq T$. By definition of sub-mutation, if a sequence chain $vs_1 \rightarrow \ldots \rightarrow vs_T$ supports M_2, it must also support M_1. So $\forall 1 \leq i \leq T, \exists r, |Support(M_2, r + i)| \leq |Support(M_1, i)|$. Therefore, $\forall 1 \leq i \leq T$, we have

$$\begin{aligned}
& mRatio(M_2, r+i) \\
= & \frac{|Support(M_2, r+i)|}{MAX_{q\in[1,k']}(count(s'_{r+i}(q), p'+q-1))} \\
\leq & \frac{|Support(M_2, r+i)|}{MAX_{q\in[1,k]}(count(s'_{r+i}(q+\Delta p), p+q-1))} \\
\leq & \frac{Support(M_1, i)|}{MAX_{q\in[1,k]}(count(s_i(q), p+q-1))} \\
= & mRatio(M_1, i)
\end{aligned}$$

By Definition 5,

$$\begin{aligned}
& mIndex(M_2) \\
= & min\{mRatio(M_2, 1), \ldots, mRatio(M_2, T')\} \\
\leq & min\{mRatio(M_2, 1+r), \ldots, mRatio(M_2, T+r)\} \\
\leq & min\{mRatio(M_1, 1), \ldots, mRatio(M_1, T)\} \\
= & mIndex(M_1)
\end{aligned}$$

□

Given a mutation index threshold $minIndex$ and a support threshold $minSup$, a k-mutation chain M is **valid** if and only if $mIndex(M) \geq minIndex$, and $Support(M) \geq minSup$.

In biology, translation probability matrix [37] is utilized to estimate the future composition of amino acids. The valid point mutations are related to the translation probability matrix because they can be directly derived from the elements of high probability in the translation probability matrix. Further, the valid k-mutation chains are the extension of valid point mutations by identifying the continuous mutated amino acid sequence positions and exploring the times of mutation. Therefore, the valid k-mutation chains present more knowledge on amino acid mutations than the valid point mutations (or translation probability matrix).

k-Mutation Chain mining Problem (k-MCP). *Given a biological sequence database BSD, minimal support $minSup$, minimal mutation index threshold $minIndex$, minimal mutation length min_k, we want to find all k-mutation chains M, where $k \geq min\_k$, $mIndex(M) \geq minIndex$ and $Support(M) \geq minSup$.*

3.3 Mining Mutation Chains

Next, we present our approach that integrates the search for the largest *k*-mutations in sequences with the discovery of chains of *k*-mutations. We first produce sequence pairs that satisfy the likelihood requirement for mutation.

3.3.1 Generate Valid Point Mutations

We can find point mutations naively by enumerating the character combinations and checking the validity of each combination via a database scan. To avoid unnecessary candidate generation, we construct a PointMutation tree to maintain the candidate point mutations of each position.

The PointMutation tree has three levels. The root node has L entries, where L is the sequence length. Each entry p points to a child node that has m entries corresponding to the m characters that can occur at position p, $1 \leq p \leq L$, $1 \leq m \leq |\Sigma|$, where Σ is the alphabet of database. Each entry consists of a character c and a pointer to a leaf node. Each leaf node has a set of entries, each of which corresponds to a mutation from c to c' at position p, as well as the set of sequence pairs that support this mutation.

We first initialize a PointMutation tree with a root node, L non-leaf nodes and their corresponding leaf nodes. The tree can be completed by scanning the set of neighboring

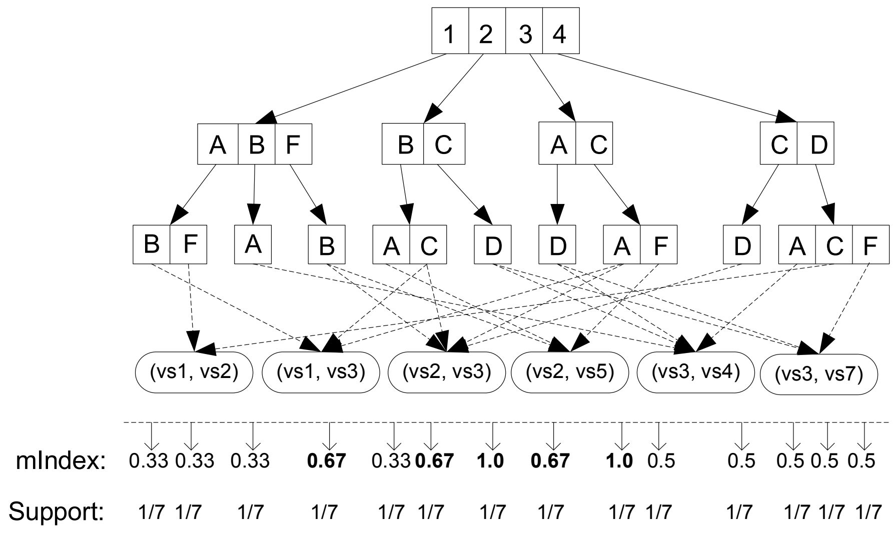

1 2 3 4
A B F
B C
A C
C D
B F
A
B
A C
D
D
A F
D
A C F
(vs1, vs2)
(vs1, vs3)
(vs2, vs3)
(vs2, vs5)
(vs3, vs4)
(vs3, vs7)
mIndex: 0.33 0.33 0.33 0.67 0.33 0.67 1.0 0.67 1.0 0.5 0.5 0.5 0.5 0.5
Support: 1/7 1/7 1/7 1/7 1/7 1/7 1/7 1/7 1/7 1/7 1/7 1/7 1/7 1/7

Figure 3.3: PointMutation tree for Figure 3.1

sequence pairs $\mathcal{SF}^2$ once. For each sequence pair $(vs_i, vs_j) \in \mathcal{SF}^2$, we extract the characters at position p, denoted as $(vs_i(p), vs_j(p))$, and create an entry for the character $vs_i(p)$ in the p-th non-leaf node if it does not already exist. We also create an entry in the leaf node for the character $vs_j(p)$ if it does not already exist and $vs_j(p) \neq vs_i(p)$, and link the leaf node to (vs_i, vs_j).

Figure 3.3 shows the PointMutation tree for our example in Figure 3.1. The root node has 4 children since the virus sequence has length 4. The first child node has 3 entries corresponding to the characters A, B, F that occur at position 1 in the virus sequence database. The entry corresponding to character A has 2 leaf nodes since A at position 1 can mutate to B and F only. The entry corresponding to character B has only 1 leaf node of A, though B at position 1 can mutate to A and B. The mutation index and support for each candidate point mutation are given at the bottom of Figure 3.3.

The PointMutation tree checks the validity of the 1-mutations as follows. During the generation of neighboring sequence pairs, we determine the occurrences of each character c at a position p, denoted by $Count(c, p)$. The support sequence pairs of a point mutation are obtained by the links of its corresponding leaf node. The mutation ratio and support are computed from $Count(c, p)$ and support sequence pairs. A point mutation is valid if its mutation ratio $\geq minIndex$ and its support is $\geq minSup$. If $minSup = 0.1$ and $minIndex = 0.6$, we have 5 valid point mutations in Figure 3.3: $\langle 1, F \rightarrow B\rangle$, $\langle 2, B \rightarrow C\rangle$, $\langle 2, C \rightarrow D\rangle$, $\langle 3, A \rightarrow D\rangle$ and $\langle 3, C \rightarrow A\rangle$.

3.3.2 Level-wise Mining

The valid point mutations need to be extended to valid k-mutations by level-wise mining, which is similar to the paradigm of frequent itemset mining [1]. Level-wise mining approach consists of two phases. The first phase generates the candidate (k+1)-mutations based on the existing valid k-mutations. The second phase evaluates the candidate (k+1)-mutations. The algorithm iterates the two phases until no valid k-mutations. Algorithm 1 gives the outline of level-wise mining. Candidate 2-mutations are generated from the valid 1-mutations if their positions are consecutive. Those 2-mutations, whose mutation ratios are no less than the *min_Index* and supports are no less than *min_sup*, are inserted into the valid 2-mutation set.

Subsequently, two valid k-mutations
$\langle p, c_p c_{p+1} \ldots c_{p+k-1} \rightarrow c'_p c'_{p+1} \ldots c'_{p+k-1}\rangle$ and
$\langle p+1, c_{p+1} \ldots c_{p+k-1} c_{p+k} \rightarrow c'_{p+1} \ldots c'_{p+k-1} c'_{p+k}\rangle$
can be joined using the union operation to form a candidate $(k+1)$-mutation
$\langle p, c_p c_{p+1} \ldots c_{p+k-1} c_{p+k} \rightarrow c'_p c'_{p+1} \ldots c'_{p+k-1} c'_{p+k}\rangle$.
Again, the mutation ratios of the newly generated $(k+1)$-mutations are computed to determine their validity. The mining process will stop when the set of valid $(k+1)$-mutations is empty.

Example 1. *We continue the mining process of running example. We have five valid 1-mutations (shown in Figure 3.4), in which one 1-mutation are in the first position, two are in the second position, two are in the third position. There is no 1-mutation in*

Algorithm 1: Level-Wise-Miner

```
input  : $\mathcal{M}_1$: the 1-mutations;
         $min\_sup$: the minimal support;
         $min\_Index$: the minimal mutation index;
         $min\_k$: the minimal length of sequence mutation.
output: the sequence mutations of length $\geq min\_k$
1  $k = 2$; $\mathcal{M} = \emptyset$;
2  while $\mathcal{M}_{k-1} \neq \emptyset$ do
3      if $k \geq min\_k$ then
4          $\mathcal{M} = \mathcal{M} \cup \mathcal{M}_{k-1}$;
5      $\mathcal{M}_k = \emptyset$;
6      foreach $(k-1)$-mutation pair $M_1$, $M_2 \in \mathcal{M}_{k-1}$ do
7          if $M_1$ and $M_2$ share a common $k-2$ point mutations then
8              $M$ = Union($M_1$,$M_2$);
9              insert $M$ into $\mathcal{M}_{cand}$;
10     $\mathcal{M}_k$ = Evaluate($\mathcal{M}_{cand}, min\_sup, min\_Index$);
11     $k++$;
12 return $\mathcal{M}$.
```

*the fourth position. The mining process is shown in Figure 3.4. To mine the longer sequence mutations, we combine the 1-mutations set such that six candidate 2-mutations are obtained. For the candidate 2-mutation $M = \langle 1, FB \rightarrow BC \rangle$, only one instance ($vs_2, vs_3$)=(**FB**CC, **BC**AD) supports M, i.e., $Support(M, t_1) = \{vs_2\}$ and $Support(M, t_2) = \{vs_3\}$. Therefore, $mIndex(M)$=$min(\frac{1}{max\{1,3\}}, \frac{1}{max\{3,1\}}) = 0.33$. Three out of six candidates have 0.0 mutation index values because no sequence pair supports them. Given min_mIndex =0.6, two candidates $\langle 2, BC \rightarrow CA \rangle$ and $\langle 2, CA \rightarrow DD \rangle$ are valid 2-mutations as their mutation index values are greater than min_mIndex.*

The level-wise mining algorithm generates candidate $(k+1)$-mutations from valid k-mutations. This may lead to the generation of many candidate mutations that do not even occur in the sequence database. For example, the 2-mutation $\langle 1, FC \rightarrow BD \rangle$ in Figure 3.4 is generated from two 1-mutations, but it is not supported by any neighboring sequence pairs. The top-down mining approach aims to overcome this drawback.

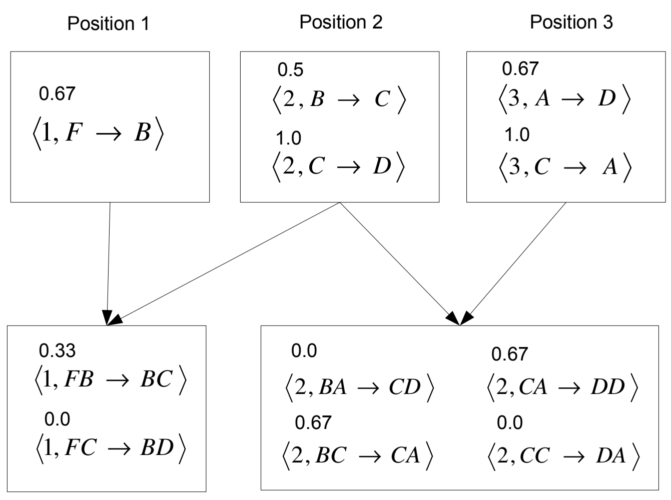

Position 1
Position 2
Position 3
0.67
$\langle 1, F \to B \rangle$
0.5
$\langle 2, B \to C \rangle$
1.0
$\langle 2, C \to D \rangle$
0.67
$\langle 3, A \to D \rangle$
1.0
$\langle 3, C \to A \rangle$
0.33
$\langle 1, FB \to BC \rangle$
0.0
$\langle 1, FC \to BD \rangle$
0.0
$\langle 2, BA \to CD \rangle$
0.67
$\langle 2, CA \to DD \rangle$
0.67
$\langle 2, BC \to CA \rangle$
0.0
$\langle 2, CC \to DA \rangle$

Figure 3.4: The mutation lattice of level-wise mining

3.3.3 Top-down Mining

Before we describe the extension of valid point mutations to valid k-mutations by top-down mining, we introduce a set of operations for the k-mutation mining phase.

- **Range(M)**. Given a k-mutation pattern M, the range operation will return the mutated position range $[p, p + k - 1]$ where p is the start position of M. We denote the lower and upper boundaries to be $R_l(M) = p$ and $R_u(M) = p + k - 1$, respectively.

- **Union(M_1, M_2)**. Given two mutation patterns, the union operation returns position-specific expressions using the *ordered* alternation symbols in regular expression for positions in Range(M_1) ∪ Range(M_2). Specifically, if we have two point mutations M_1=$\langle p, c_1 \to c_1' \rangle$ and M_2=$\langle p, c_2 \to c_2' \rangle$, then Union($M_1, M_2$) is given by $\langle p, [c_1|c_2] \to [c_1'|c_2'] \rangle$. For example, $\langle 2, [a|b] \to [c|d] \rangle$ means that at position 2, a can mutate to c, or b can mutate to d, but **not** a to d, nor b to c.

- **Intersect(M_1, M_2)**. Given two mutation patterns, the intersect operation will return the common character for each position in Range(M_1) ∩ Range(M_2). Hence if we have two point mutations M_1=$\langle p$,c_1→$c'_1\rangle$ and M_2=$\langle p$,c_2→$c'_2\rangle$, the intersect at position p is given by

$$\begin{cases} < p, [c] \to [c'] > \text{ if } c1 = c2 = c \text{ and } c'_1 = c'_2 = c' \\ < p, [\epsilon] \to [c'] > \text{ if } c1 \neq c2 \text{ and } c'_1 = c'_2 = c' \\ < p, [c] \to [\epsilon] > \text{ if } c1 = c2 = c \text{ and } c'_1 \neq c'_2 \\ \quad \epsilon \qquad\qquad \text{otherwise} \end{cases}$$

In Section 3.3.1, we generate all the valid 1-mutations. We can compact these 1-mutations into position-specific expressions to form a maximal mutation expression, $M_{max} = (R_1 \to R_2)$, where R_1 and R_2 are regular expressions obtained by concatenating all the compact expressions at each position p, $1 \leq p \leq L$, L is the sequence length. For example, by combining all valid point mutations in Figure 3.3, we have M_{max} = $\langle 1, F[B|C][A|C]\epsilon \to B[C|D][D|A]\epsilon\rangle$.

All candidate mutations can be obtained by scanning the sequence pairs once. For each sequence pair (vs_i, vs_j), we generate a *hit mutation* by Intersect(M_{max}, M), where M=$\langle 1$,vs_i→$vs_j\rangle$. For example, given $M_{max} = \langle 1, F[B|C][A|C]\epsilon \to B[C|D][D|A]\epsilon\rangle$, the hit mutation for sequence pair (ABCD, FBCC) is $\langle 1$,$\epsilon BC\epsilon$→$\epsilon\epsilon\epsilon\epsilon\rangle = \epsilon$, while the hit mutation for sequence pair (FBCC, BCAD) is $\langle 1$,$FBC\epsilon$→$BCA\epsilon\rangle$ = $\langle 1$,FBC→$BCA\rangle$.

Non-empty hit mutations will be stored and their support sequence pairs set updated to facilitate the subsequent mining process. We design a tree called *MaxMutation tree* to store these information. In MaxMutation tree, the root node is the maximal mutation, and the nodes are ordered based on their sub-mutation relationship:

1. A node d is the parent of a node e if the k-mutation at node e is a sub-mutation of that at node d, and

2. All the sibling nodes do not have a sub-mutation relationship with each other.

For each sequence pair (vs_i, vs_j), we obtain a hit mutation $M = \langle p, s_1 \rightarrow s_2 \rangle$ by intersecting (vs_i, vs_j) with the maximal mutation M_{max} and ignoring positions with empty mutations ϵ. We perform a breadth-first traversal of the *MaxMutation* tree to insert the hit mutation. Let d be a node that contains a k-mutation $M_d = \langle p', s_1^d \rightarrow s_2^d \rangle$.

1. If $p = p'$ and $s_1 = s_1^d$ and $s_2 = s_2^d$, then insert the sequence pair (vs_i, vs_j) into the support set of d.

2. If $p \geq p'$ and s_1 is a substring of s_1^d starting at position p and s_2 is a substring of s_2^d starting at position p, that is, M is a sub-mutation of M_d, then we create a new node *nd* to store the k-mutation $\langle p, s_1 \rightarrow s_2 \rangle$ and attach *nd* as a child node of d. The support set in *nd* is the sequence pair (vs_i, vs_j).

3. If $p \leq p'$ and and s_1^d is a substring of s_1 starting at position p' and s_2^d is a substring of s_2 starting at position p', that is, M_d is a sub-mutation of M, then we create a new node *nd* to store the k-mutation $\langle p, s_1 \rightarrow s_2 \rangle$ and insert *nd* between d and d's parent. The support set in *nd* is the sequence pair (vs_i, vs_j).

4. Otherwise, create a new node *nd* to store the k-mutation and insert it as a child of the root node. The support set in *nd* is the sequence pair (vs_i, vs_j).

The above enumeration will miss implicit sub-mutations. Suppose the *MaxMutation* tree has nodes $M_1 = \langle 1, ABCD \rightarrow EFGH \rangle$ and $M_2 = \langle 1, RBCT \rightarrow JFGW \rangle$, which imply a sub-mutation $M_{sub} = \langle 2, BC \rightarrow FG \rangle$. To ensure completeness of the patterns mined, we need to explicitly store all such sub-mutations by checking for intersections between the k-mutation $\langle p, s_1 \rightarrow s_2 \rangle$ and the mutations in the *MaxMutation* tree. If we find any non-empty intersection with a mutation at a node d, a new hit mutation is obtained and inserted into the *MaxMutation* tree as described above.

Algorithm 2 gives the details of the construction process. Let M_{hit} be the hit mutation of sequence pair (vs_i, vs_j) and M_{max}. Lines 1-2 find d's child nodes that contain

Algorithm 2: InsertHitMutation(M_{hit}, (vs_i, vs_j), d)

```
input : $M_{hit}$: the hit mutation to be inserted;
        $(vs_i, vs_j)$: the support sequence pair;
        $d$: the current node;
1  $C = \{d_c | d_c$ is a child node of $d \wedge M_{d_c} \sqsubseteq M_{hit}\}$ ;
2  $\mathcal{P} = \{d_p | d_p$ is a child node of $d \wedge M_{hit} \sqsubseteq M_{d_p}\}$;
3  $O = \{d_o | d_o$ is a child node of $d \wedge d_o \notin C \wedge d_o \notin \mathcal{P}\}$;
4  if $O \neq \emptyset$ then
5      foreach $d_o \in O$ do
6          $M'_{hit}$= Intersect($M_{hit}$, $M_{d_o}$);
7          InsertHitMutation($M'_{hit}, (vs_i, vs_j), d_o$);
8  if $\exists\, d_e$, such that $d_e$ is a child of $d \wedge M_{d_e} = M$ then
9      Insert $(vs_i, vs_j)$ into $d_e$'s support;
10 else if $C \neq \emptyset$ then
11     Remove all nodes in $C$ from $d$'s child nodes ;
12     Create a node $n_w$ of mutation $M_{hit}$ as a child node of $d$;
13     Attach all nodes in $C$ as the child nodes of $n_w$;
14 else if $\mathcal{P} \neq \emptyset$ then
15     foreach $d_p \in \mathcal{P}$ do
16         InsertHitMutation($M_{hit}, (vs_i, vs_j), d_p$);
17 else
18     Create a node $n_w$ of mutation $M_{hit}$ as a child node of $d$;
```

sub-mutations and super-mutations of M_{hit}. Line 3 finds d's child nodes other than the nodes in C and $\mathcal{P}$. Lines 4-7 recursively intersect M_{hit} and each node d_o in O, and insert the intersection mutations into d_o. Lines 8-9 insert the sequence pair (vs_i, vs_j) into the support set of node d_e, if d_e contains the mutation equal to M_{hit}; otherwise, if node d has a set of child nodes C that contains sub-mutations of M_{hit}, we create a new node n_w to store M_{hit} and insert n_w between node d and the nodes in C (Lines 11-13). If node n has an empty child node set C but a non-empty $\mathcal{P}$, we call Algorithm 2 to insert M_{hit} into each parent node in $\mathcal{P}$ (Lines 15-16). If both C and $\mathcal{P}$ are empty, we create a new node n_w of mutation M_{hit} as a child node of d (Line 18).

Lemma 2. *The* MaxMutation *tree stores the complete set of candidate k-mutations.*

Algorithm 3: Top-Down-Miner

```
input  : $\mathcal{M}_1$: the 1-mutations;
         $minSup$: the minimal support;
         $minIndex$: the minimal mutation index;
         $min\_k$: the minimal mutation length;
output: the maximal $k$-mutations, where $k \geq min\_k$
1  $M_{max}$ = Union($\mathcal{M}_1$) ;
2  initialize the node $root$ of mutation tree of $M_{max}$;
3  foreach $sequence\ pair(vs_i,vs_j) \in \mathcal{SF}^2$ do
4      $M_{hit}$ = Intersect($(vs_i,vs_j)$, $M_{max}$) ;
5      InsertHitMutation($M_{hit}, (vs_i, vs_j), root$);
6  $\mathcal{M}_k = \emptyset$;
7  TopdownEvaluate($MTree.root, minSup, minIndex, \mathcal{M}_k$);
8  return $\mathcal{M}_k$.
9  Procedure TopdownEvaluate($d$, $minSup$, $minIndex$, $\mathcal{M}_k$)
10 if $\exists M \in \mathcal{M}_k$ such that $M_d \sqsubseteq M$ then
11     return;
12 if $mIndex(M_d) \geq minIndex \wedge Support(M_d) \geq minSup$ then
13     insert $M_d$ into $\mathcal{M}_k$;
14 else
15     foreach child node $d_c$ of $d$ do
16         $Support(d_c) = Support(d_c) \cup Support(d)$;
17         TopdownEvaluate ($d_c$, $minSup$, $minIndex$, $\mathcal{M}_k$);
```

Proof: Let the hits be generated in the order $\mathcal{H}_1$, $\mathcal{H}_2$, …, $\mathcal{H}_n$. We prove by induction. When i=1, all candidate mutations from $\mathcal{H}_1$ are direct children of the root node. When $i = 2$, let $SetH_1$ and $SetH_2$ be the sets of candidate mutations from $\mathcal{H}_1$ and $\mathcal{H}_2$ respectively. For each pair of k-mutations h_1 and h_2, $h_1 \in SetH_1$, $h_2 \in SetH_2$, one of these cases is true:

1. There exists a sub-mutation relationship between h_1 and h_2. Without loss of generality, let h_1 be a sub-mutation of h_2. Then h_1 will be a child node of h_2 in the tree.

2. There is an overlap between h_1 and h_2. In this case, the overlap must be a sub-mutation of h_1 and h_2 and is inserted as child node of both h_1 and h_2.

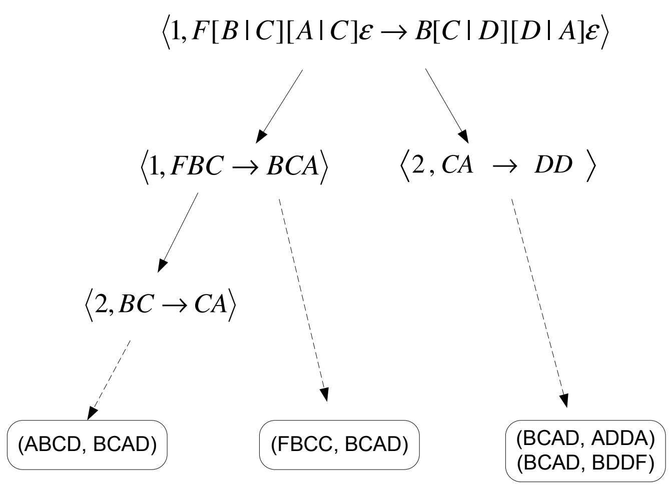
3. There is no overlap between h_1 and h_2. Both will be inserted as child nodes of the root.

Thus, the lemma is true for the base cases since all candidate mutations for $\mathcal{H}_1$ and $\mathcal{H}_2$ are kept in the tree. Assume that the *MaxMutation* tree is complete for $i = m$. When $i = m + 1$, if the k-mutations of $\mathcal{H}_i$ overlap with the mutations of the previous hits $d_{\mathcal{H}_{j1}}$, $d_{\mathcal{H}_{j2}}, \ldots, d_{\mathcal{H}_{jk}}$, where $1 \leq j1 \leq \ldots \leq jk \leq i$, the overlap will be stored in some nodes created along the path $d_{\mathcal{H}_{j1}}, d_{\mathcal{H}_{j2}}, \ldots, d_{\mathcal{H}_{jk}}$ starting from the root. Hence, no candidate mutations will be missed.

□

Algorithm 3 shows the top-down mining algorithm. We first construct the *MaxMutation* tree to capture the k-mutations and their support sets (Lines 1-5). Next, we initialize a variable $\mathcal{M}_k$ to maintain the list of valid mutations (Line 6). Each node in the *MaxMutation* tree is associated with an instance set that stores the support sets of its ancestor nodes, including its own support set. This instance set is updated as we traverse the tree in a depth-first manner. The instance set of a node d is used to compute the *mIndex* of the corresponding k-mutation. If the k-mutation at node d is valid, we insert it into $\mathcal{M}_k$, and we do not visit the child nodes of d. Finally, $\mathcal{M}_k$ contains the list of valid k-mutations (Lines 9-17).

From Figure 3.3, we have $M_{max} = \langle 1, F[B|C][A|C]\epsilon \rightarrow B[C|D][A|D]\epsilon\rangle$ by combining the valid 1-mutations at all of the four positions. We generate the hit mutations for the six virus pairs in Figure 3.1, four of which satisfy the $min\_k = 2$. These hit mutations, $\langle 1, \epsilon BC\epsilon \rightarrow \epsilon CA\epsilon\rangle$, $\langle 1, FBC\epsilon \rightarrow BCA\epsilon\rangle$, $\langle 1, \epsilon CA\epsilon \rightarrow \epsilon DD\epsilon\rangle$, $\langle 1, \epsilon CA\epsilon \rightarrow \epsilon DD\epsilon\rangle$, are translated into the k-mutations by removing the empty mutation ϵ: $\langle 2, BC \rightarrow CA\rangle$, $\langle 1, FBC \rightarrow BCA\rangle$, $\langle 2, CA \rightarrow DD\rangle$, $\langle 2, CA \rightarrow DD\rangle$, and inserted into the *MaxMutation* tree. Figure 3.5 shows the final *MaxMutation* tree obtained. If *minIndex*=0.6, then the candidate mutation $\langle 1, FBC \rightarrow BCA\rangle$ is not valid. Its support is propagated to its child node $M_c = \langle 2, BC \rightarrow$



$\langle 1, F[B \mid C][A \mid C]\varepsilon \to B[C \mid D][D \mid A]\varepsilon \rangle$
$\langle 1, FBC \to BCA \rangle$
$\langle 2, CA \to DD \rangle$
$\langle 2, BC \to CA \rangle$
(ABCD, BCAD)
(FBCC, BCAD)
(BCAD, ADDA)
(BCAD, BDDF)

Figure 3.5: MaxMutation tree for Figure 3.1

$CA\rangle$, which leads to $Support(M_c, 1) = \{ABCD, FBCC\}$ and $Support\,(M_c, 2) = \{BCAD\}$, such that $mIndex(M_c) = 0.67$. Note that $\langle 2, BC{\to}CA\rangle$ and $\langle 2, CA{\to}DD\rangle$ are valid maximal mutations.

We can further improve the Top-Down-Miner by applying a *position pruning strategy* which is based on the observation that certain positions cannot have any valid mutations.

Lemma 3. *Let minSup be the minimal support. Sequences corresponding to positions* $[p_i, \cdots, p_{i+k-1}]$ *can support a valid k-mutation chain with t states if and only if the entropy measure corresponding to these positions lies in the range* $[H_{lb}, H_{ub}]$, *where* $H_{lb} = \log t$ *and* $H_{ub} = -t \cdot minSup \cdot \log minSup + (1 - t \cdot minSup) \log |D|]$, *D is the total number of sequences.*

PROOF: Entropy measures the purity of an attribute. A low entropy measure implies high purity. Positions $[p_i, \cdots, p_{i+k-1}]$ have the lowest entropy when we have exactly t subsequences and these t subsequences have identical frequencies. For this case, the entropy is given by

$$\begin{aligned} H_{lb} &= \sum_{i=1}^{t} -\frac{1}{t} \log \frac{1}{t} \\ &= \log t \end{aligned}$$

The worst case occurs when positions $[p_i, \cdots, p_{i+k-1}]$ have t subsequences each having a frequency of $minSup$, with $(1 - t \cdot minSup)|D|$ number of subsequences being fully random. Let $\alpha = (1 - t \cdot minSup)$.

$$\begin{aligned} H_{ub} &= -\sum_{i=1}^{t} minSup \cdot \log minSup - \sum_{i=1}^{\alpha|D|} \frac{1}{|D|} \log \frac{1}{|D|} \\ &= -t \cdot minSup \cdot \log minSup + \alpha \log |D| \end{aligned}$$

This corresponds to the entropy bounds of $[H_{lb}, H_{ub}]$.

□

With this lemma, we can prune a position if there is no sequence involving this position whose entropy measure falls in the range $[H_{lb}, H_{ub}]$.

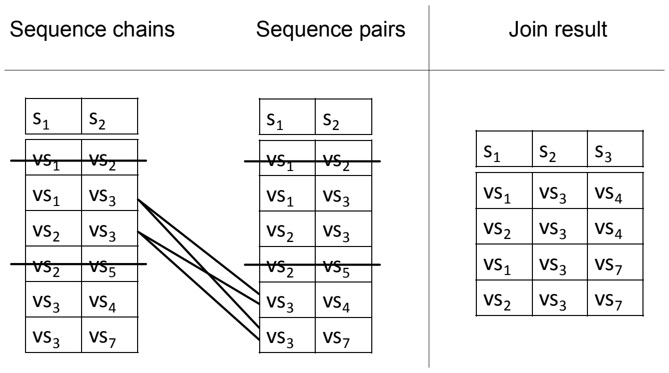


Figure 3.6: Generation of mutation chains by Selective Join

3.3.4 Generate Mutation Chains

In this section, we describe the mining of mutation chains. The naive approach is to generate a chain of k-mutation with length T from chains of k-mutation with length $T-1$. But this is infeasible as the Apriori property does not hold. For example, sequence chain (vs_1, vs_2, vs_3) supports a k-mutation chain $\langle 1, AB{\rightarrow}BB{\rightarrow}CD\rangle$, but it may not support the subsequence of this chain $\langle 1, AB{\rightarrow}CD\rangle$, as it may not satisfy the likelihood requirement for mutation. Hence, we cannot join k-mutations with length $T-1$ to obtain candidate k-mutations with length T. To tackle this problem, we define an operator called selective join, denoted as $\bowtie_s$:

$$\mathcal{SF}^T = \mathcal{SF}^{T-1} \bowtie_s \mathcal{SF}^2$$

where $\mathcal{SF}^{T-1}$ is the set of sequence chains of length $T-1$, and $\mathcal{SF}^2$ is the set of sequence pairs. A sequence chain $(vs_1, \ldots, vs_{T-1})$ in $\mathcal{SF}^{T-1}$ will join with a sequence pair (vs_i, vs_j) in $\mathcal{SF}^2$ if and only if

1. $(vs_1, \ldots, vs_{T-1})$ supports a valid k-mutation chain $(s_1 \rightarrow \ldots \rightarrow s_{T-1})$;

2. (vs_i, vs_j) supports a valid k'-mutation $(s_i \rightarrow s_j)$;

3. s_{T-1} and s_i share a common substring of length $k'' \geq min\_k$.

The selective join will greatly prune the sequence chains, and it will not miss any potential sequence chains which are possible to support one of valid k-mutation chains. This is guaranteed by Lemma 4.

Lemma 4. *If a sequence chain of length T supports a mutation chain, it supports all subsequences of the chain.*

Proof: Let $s_1 \rightarrow s_2 \rightarrow \ldots \rightarrow s_T$ to be a k-mutation chain of length T, and $vs_1 \rightarrow vs_2 \rightarrow \ldots \rightarrow vs_T$ be the virus sequence chain that supports the mutation chain. This implies vs_1 support $s_1, \ldots, vs_T$ support s_T. Then for each subsequence $s_i \rightarrow \ldots \rightarrow s_j$, where $1 \leq i \leq j \leq T$, there exists a corresponding subsequence, $vs_i \rightarrow \ldots \rightarrow vs_j$, such that vs_i support $s_i, \ldots, vs_j$ support s_j.

□

Figure 3.6 shows the process of selective join. We first delete the sequence pairs (vs_1, vs_2) and (vs_2, vs_5) because they do not support any valid k-mutations $\langle 2, BC \rightarrow CA\rangle$ and $\langle 2, CA \rightarrow DD\rangle$. The remaining sequence pairs (vs_1, vs_3) and (vs_3, vs_4) can be joined because (vs_1, vs_3) supports $\langle 2, BC \rightarrow CA\rangle$ and (vs_3, vs_4) supports $\langle 2, CA \rightarrow BD\rangle$, and the last state of $\langle 2, BC \rightarrow CA\rangle$ share a common "CA" with the first state of $\langle 2, CA \rightarrow BD\rangle$. Finally, we obtain four sequence chains. Note that two sequence chains (vs_1, vs_2, vs_3) and (vs_1, vs_2, vs_5) are pruned based on Lemma 4.

Algorithm 4 shows the k-Mutation-Miner (kMM) framework to mine k-mutation chains. Lines 1-2 first initialize an empty mutation set $\mathcal{M}$, generate the sequence pairs $\mathcal{SF}^2$. Line 3 prunes the positions which are impossible to contain valid k-mutations by entropy bounds, and Line 4 finds the valid point mutations on the unpruned positions. Line 5 mines the valid k-mutations of length 2 by using the Top-Down-Miner. The k-mutation chains are stored in $\mathcal{M}$ (Line 6). Next, we generate valid k-mutations of increasing length t by applying the selective join operator on $\mathcal{SF}^t$ and $\mathcal{SF}^2$, $t \geq 2$ to obtain $\mathcal{SF}^{t+1}$. With $\mathcal{SF}^{t+1}$, we call Top-Down-Miner to discover the valid k-mutation chains of length $t + 1$ using a modified *MaxMutation* tree. The nodes in the modified *MaxMutation* tree stores k-mutation chains instead of k-mutations. The process continues until $\mathcal{M}_t$ is empty (Lines 8-13). Line 14 returns $\mathcal{M}$.

Theorem 1. *The k-mutation chains returned by Algorithm 4 is correct and complete.*

Proof: For correctness, we prove that both the sequence pairs and sequence chains all satisfy the likelihood requirement for mutation. Step 2 of Algorithm 4 ensures that only the sequence pairs satisfying the likelihood requirement for mutation will par-

Algorithm 4: *k*MM

```
input  : $BSD$: biological sequence database;
         $minSup$: the minimum support;
         $minIndex$: the minimum mutation index;
         $min\_k$: the minimum mutation length.
output:  the maximal $k$-mutation chains, where $k \geq min\_k$
1  $\mathcal{M} = \emptyset$;
2  $\mathcal{SF}^2$ = set of sequence pairs that satisfies the likelihood requirement for
   mutation;
3  Perform position pruning strategy;
4  Find valid point mutations $\mathcal{M}_1^2$ from $\mathcal{SF}^2$ (Section 3.3.1);
5  $\mathcal{M}_K^2$ = Top-Down-Miner ($\mathcal{M}_1^2$, $minSup$, $minIndex$, $min\_k$) (Section 3.3.3);
6  $\mathcal{M} = \mathcal{M} \cup \mathcal{M}_K^2$;
7  $t = 2$;
8  while $\mathcal{M}_K^t \neq \emptyset$ do
9      $\mathcal{SF}^{t+1} = \mathcal{SF}^t \bowtie_s \mathcal{SF}^2$ (Section 3.3.4);
10     Perform position pruning strategy;
11     Find point mutation chains $\mathcal{M}_1^{t+1}$ from $\mathcal{SF}^{t+1}$ (Section 3.3.1);
12     $\mathcal{M}_K^{t+1}$ = Top-Down-Miner ($\mathcal{M}_1^{t+1}$, $minSup$, $minIndex$, $min\_k$) (Section 3.3.3);
13     $\mathcal{M} = \mathcal{M} \cup \mathcal{M}_K^t$; $t++$;
14 return $\mathcal{M}$ ;
```

ticipate in the selective join. Since sequence chains of length T are obtained by the selective join between chains of length $T-1$ with the sequence pairs obtained in Step 2, the sequence chains generated also satisfy the likelihood requirement for mutation.

For completeness, Lemma 2 states that the MaxMutation tree maintains the complete set of candidate k-mutation chain. We use a counter example to prove that the sequence chains set $\mathcal{SF}^T$ is complete for $T > 2$. Let $M' \sqsubseteq M$ and M is a valid k-mutation chain of length T and M' is a k-mutation chain of $T-1$. From the anti-monotonicity property (Lemma 1), M' is a valid k-mutation chain. Let a sequence chain (vs_1, ..., vs_T) support M. If the sequence chain is excluded in $\mathcal{SF}^T$, it does not support any valid k-mutation chain of length $T-1$ according to the operation of selective join. However, (vs_1, ..., vs_T) supports M' according to Lemma 4. This is a contradiction.

□

3.4 Experimental Studies

In this section, we report the results of our mining algorithm on both synthetic and real world datasets. All the algorithms are implemented in C++ and the experiments are carried out on a server with dual Xeon 3GHZ processors and 4GB memory, running Windows server 2003.

3.4.1 Experiments on Synthetic Datasets

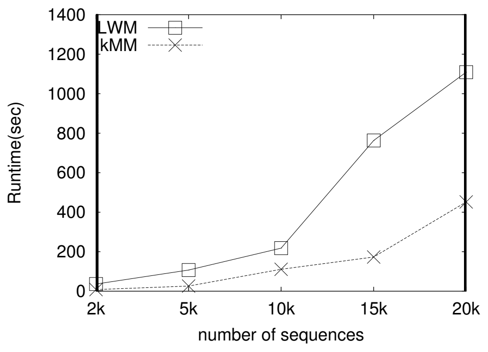
The synthetic datasets are generated by modifying the data generator in [33]. We use two parameters to generate sequences with location and time: L is the length of sequences and min_k is the length of mutations. By default, the space and time dimensions are set to $1000 \times 1000 \times 1200$, and the alphabet size $|\Sigma|$ is set to 20. We use the notation $DATA-(|\mathcal{D}|)-(L)$ to denote a dataset of $|\mathcal{D}|$ sequences and L sequence length.

We also develop the level-wise-miner (LWM) as a baseline for comparison. LWM generates candidate (k+1)-mutations based on existing valid k-mutations and evaluates candidate (k+1)-mutations to find valid mutations whose mutation ratios and supports are no less than the $minIndex$ and $minSup$ respectively.

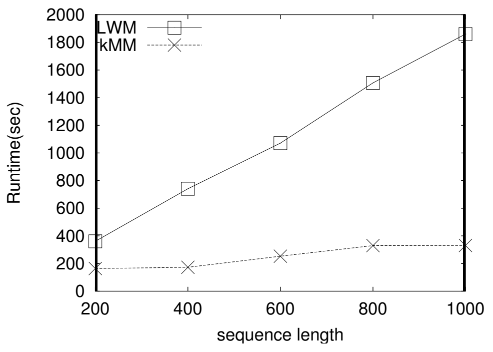
LWM V.S. *k*MM

We examine the scalability of LWM and *k*MM by varying the database size $|\mathcal{D}|$ and the sequence length L. We incorporate three k-mutation chains of length T, where $7 \leq k \leq 14$ and $2 \leq T \leq 5$, into the datasets.

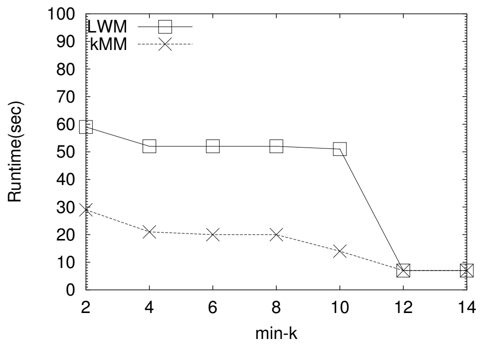
Figure 3.7(a) shows the results for varying $|\mathcal{D}|$ and L=100. We observe that LWM is much slower than *k*MM. This is expected the *k*MM detect mutations in the top-down manner, which is more efficient to find long k-mutations. Figure 3.7(b) shows the results for varying L. We see that LWM is slower than the *k*MM by an order of magnitude as L increases.



(a) Effect of database size



(b) Effect of sequence length

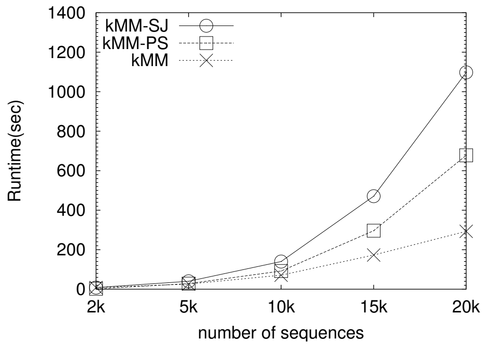


(c) Effect of *min_k*

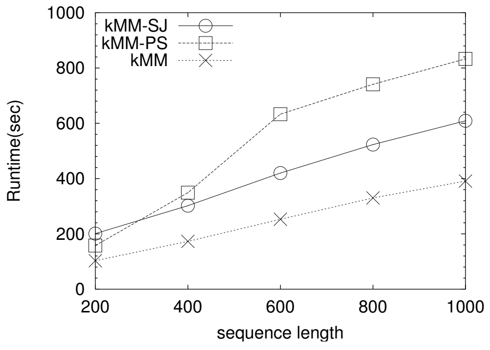
Figure 3.7: Comparative study of *k*MM and LWM

We also test the performance of LWM and *k*MM by varying *min_k*. The dataset used has 5,000 sequences and contains three *k*-mutation chains of length 2, where 5≤*k*≤10. Figure 3.7(c) shows that runtime of LWM and *k*MM suddenly decrease at *min_k*=12. This is because both algorithms terminates as they do not find any valid *k*-mutations when *min_k*≥12.

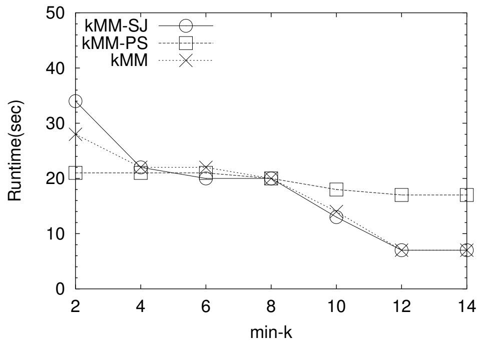
We observe that *min_k* has no salient effect on LWM when *min_k*≤10. This is because LWM perform level-wise mining which combine the point mutations at the final stage, hence it cannot prune infeasible point mutations early. Figure 3.7(c) also shows that that runtime of *k*MM decrease as *min_k* increases. This is because a large *min_k* reduces the number of hit mutations in MaxMutation tree, which leads to less time in constructing and traversing the *MaxMutation* tree.



(a) Effect of database size



(b) Effect of sequence length



(c) Effect of *min_k*

Figure 3.8: Effect of pruning techniques

Effect of pruning techniques in *k*MM

Next, we examine the effectiveness of the pruning techniques, position selection and selective join. We have two variants of *k*MM: *k*MM-PS which is *k*MM without position selection and *k*MM-SJ which is *k*MM without selective join. In *k*MM-SJ, we join all instances to obtain instance chains.

We fix sequence length L=100 and generate datasets by varying $|\mathcal{D}|$ from 2k to 20k and incorporate three k-mutation chains of length T, where $7 \leq k \leq 14$ and $2 \leq T \leq 4$, into the datasets. We set *min_k*=7. The results in Figure 3.8(a) shows that *k*MM outperforms *k*MM-SJ because *k*MM prunes the sequence instances which do not support any valid mutations using selective join. Figure 3.8(a) also show that *k*MM outperforms *k*MM-PS. This is because *k*MM prunes the positions which will not satisfy the entropy bound.

We also study the effect of sequence length L. Each dataset has three k-mutation chains of length T, where $7 \leq k \leq 14$ and $2 \leq T \leq 4$. We set *min_k*=7. Figure 3.8(b) shows the results. We observe that *k*MM is faster than both *k*MM-PS and *k*MM-SJ because *k*MM can prune more positions as the sequence length increases, and prune the sequence instances.

Finally, we test the effect of varying *min_k*. In this experiment, the dataset has 5,000 sequences and contains three *k*-mutation chains of length 2, where $5 \leq k \leq 10$. Figure 3.8(c) shows the results. We see that the runtime of *k*MM-PS and *k*MM decrease slowly with the increase of *min_k*. This is because the number of hit mutations in MaxMutation tree is reduced as *min_k* increases, which leads to less time for constructing MaxMutation tree. We also see that the runtime of *k*MM decrease faster than that of *k*MM-PS. This is because *k*MM can prune positions.

3.4.2 Experiments on Influenza A Virus Dataset

Next, we use the influenza A virus protein dataset [4] to discover meaningful mutation chains. The dataset contains information on the sequences of 11 influenza A viral proteins, including the subtype (e.g., H5N1,H1N1), host (e.g., human, avian), country and year of isolation. Table 3.2 shows the the length and number of sequences for each subtype in influenza A dataset.

Alignment. Multiple sequence alignments of the 11 proteins (as listed in Table 3.2) were carried out with MUSCLE 3.6 [17]. Due to the great variability exhibited by the HA and NA protein, separate alignments were obtained from each subtype (16 subtypes for HA and 9 for NA). The subtype alignments were merged using the MUSCLE tool to obtain the final HA and NA alignment. The introduction of gaps in the resulting alignments was minimized by merging sequences based on sequence similarity between subtypes. The sequence lengths after alignment are shown in the “Length” column in Table 3.2.

Table 3.2: The meta data of Influenza A virus proteins dataset

Protein	Length	H5N1	H1N1	H3N2	others	Total
PB2	759	520	518	1384	721	3143
PB1	758	516	476	1402	780	3174
F2	90	58	285	1263	1646	1989
PA	718	520	419	1380	383	3111
HA	610	790	1897	3240	1189	7116
NP	499	549	405	1653	1156	3763
NA	493	715	968	1790	748	4221
M1	252	579	259	1517	1426	3781
M2	97	390	370	1491	898	3149
NS1	237	567	443	1429	1320	3759
NS2	121	374	364	1404	978	3120
Total	4634	5578	6404	17953	10391	40326

Likelihood of mutation. We observe that one protein sequence vs_1 is likely to mutate to another sequence vs_2 if vs_2 occurs within two years from the occurrence vs_1, and the geographical distance of vs_1 and vs_2 is less than 1,000 kilometers, and their edit distance is less than 20% of the whole sequence length. This makes sense because the viruses spread and mutate gradually, instead of sudden changes and promulgation.

All experiments are performed by setting minSup=0.01 and minIndex=0.5, which are suggested by our biologists.

On H5N1 subtype (bird flu)

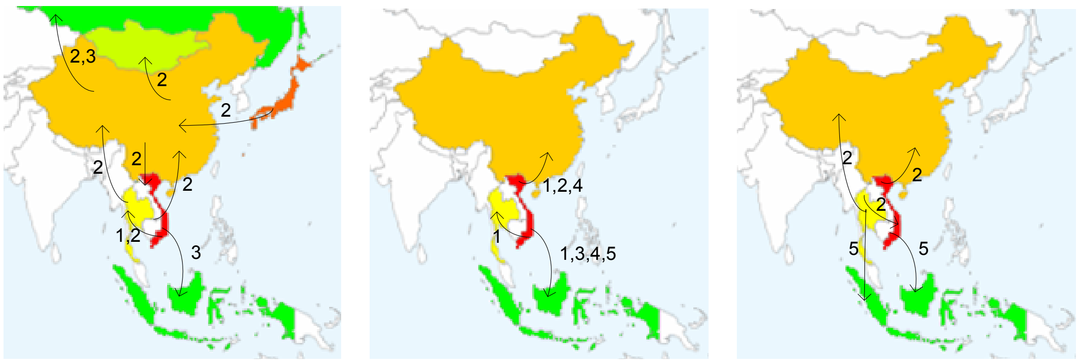
We apply our algorithm to discover the point mutations and *k*-mutations on the H5N1 subtype (bird flu). All in all, we discovered 205 point mutations using our algorithm, as shown in Table 3.3. We use the abbreviation cpc' to denote $\langle p, c \rightarrow c' \rangle$.

Within all of the point substitutions, we highlight the point mutation $\langle 627, E \rightarrow K \rangle$ (E627K for short) in protein PB2. $E627K$ has been shown to have important biological effect of converting a nonlethal H5N1 influenza A virus to a lethal virus [69]. We examine the geographical spread for $E627K$ as shown in Figure 3.9(a). The spatiotemporal

Table 3.3: The amino acid substitution in H5N1 subtype

Protein	Point Mutations
PB2	M64I, A105T, A108T, T108A, R288Q, K299R, K339T, K340R, M483V, S590G, **E627K**, I649V, V649I, A661T, I667V
PB1	A14V, V113I, V149I, K215R, R215K, S375N, S384L, R386K, T400A
F2	L37R, R69Q, R81K
PA	G58S, V100I, I129T, R204K, S245K, F246L, L247S, N248Q, V249M, I324V, A338T, G366S, I389V, V389I, K393R, R393K, D396N, P402S, I556V, P655S, A671V, T714A, R718K
HA	R67K, D77N, N77D, N116S, S116N, V118A, V118T, D126N, N157D, N157S, S157D, E159D, L162S, Q172L, K174R, K174S, P176S, S176P, S192N, A193T, T193A, M212L, K226R, R226K, V237I, S254P, D264E, E264D, A309T, L316V, V316L, M330I, K358R, R358K, T368S, R380G, D443N, K530R, K540R, N556K, M568I, I571T
NP	G34S, K77R, M136L, I183V, A354I, A374T, T374A, S483N
NA	I17T, T17I, I20V, V20I, V26I, I29M, H39Q, Q39H, H44R, P48S, N82K, A91V, V91A, N106R, R106N, S106N, I110V, V110I, H111Y, Y111H, G116S, K122R, H166Y, Y166H, G212E, Y264H, R268K, I269M, M269I, D282N, N282D, M356V, V356M, L358P, P358L, P358S, S358P, I366V, V366I, E402G, G402E, N407S, S407N, G477S, S477G
M1	V15I, R27K, T37A, I59M, I107M, F144L, L144F, T168I, S207N, S224N, K230R, D232N, N232D, I234L, L234I
M2	G14E, R18K, N31S, A64S, S82N
NS1	L22F, L27M, M27L, N48S, E55R, R55E, A60E,E60A, H63Q, Q63H, P87S, F103L, L103F, I106M, A112T, T112A, K118R, R118K, N127T, F138Y, N139D, A143T, T143A, D152E, G153E, D171G, D171N, E171D, G171N, N171G, I180V, V180I, L198I, N207D, D209N, N209D, P212L, P213S, S213P, N217K, E229K, K229E
NS2	M14V, V14M, V52M, A115T

spread patterns for $E627K$ suggests that the virus originated in Vietnam, and spread outwards and eventually caused the disease outbreaks in the two northern countries, namely Russia and Mongolia.



(a) PB2: $\langle 627, E \rightarrow K\rangle$ (b) PA: $\langle 244, SFLNV \rightarrow KLSQM\rangle$ (c) HA: $\langle 571, ---------- \rightarrow YQILSIYST\rangle$

Figure 3.9: The dominant support chains for mutations in H5N1 subtype. 1 means Year 2003-2004, 2 means Year 2004-2005, 3 means Year 2005-2006, 4 means Year 2003-2005, 5 means Year 2004-2006

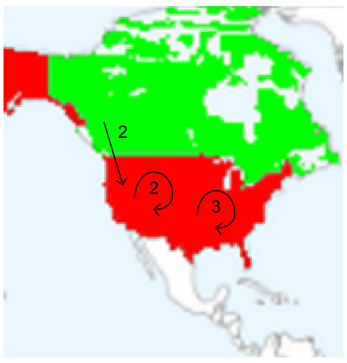
In addition to discovering point mutations, our method also discovers a 5-mutation pattern in PA protein, $\langle 244, SFLNV \rightarrow KLSQM\rangle$. Figure 3.9(b) shows the geographical spread of this mutation. We can see that the spread chains origin from Vietnam and spread to three neighboring countries, China, Indonesia and Cambodia.

We also find an insertion mutation in HA protein: $\langle 571, ---------- \rightarrow YQILSIYST\rangle$. Its dominant spread flows are shown in Figure 3.9(c).

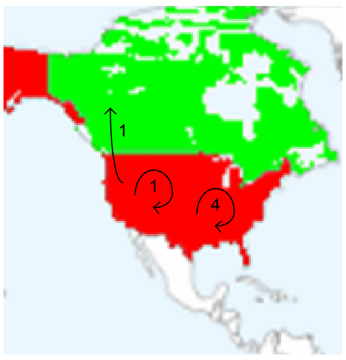
On H1N1 subtype (swine flu)

We also apply our algorithm to discover point mutations and k-mutations on the H1N1 subtype. Due to the importance of the polymerase genes (i.e., PB1 and PB2) in adaptive mutations and potential reassortment [44], we focus the mutation mining in such proteins. We discovered one 2-mutations in the PB1 protein $\langle 455, HE \rightarrow YA\rangle$, and two 2-mutation in the PB2 protein $\langle 489, NA \rightarrow ST\rangle$ and $\langle 489, ST \rightarrow NA\rangle$.

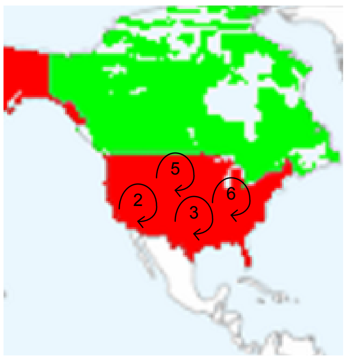
Figure 3.10(a) depicts two major geographical spread chain of PB1. The first spread occurred during 2002 and 2003, and spread from Canada to USA and an internal mutation in USA. The second spread occurred during 2005 and 2007 in USA. Similarly, Figure 3.10(b) shows two major geographical spread chain of PB2 $\langle 489, NA \to ST \rangle$. The first one occurred during 2001 and 2003, and spread from USA to Canada and include an internal mutation in USA. The second spread occurred during 2007 and 2009 in USA, indicating that this mutation is likely to have influenced the recent epidemic swine flu. Figure 3.10(c) depicts four internal spreads for the PB2 mutation $\langle 489, ST \to NA \rangle$.



(a) PB1: $\langle 455, HE \to YA \rangle$



(b) PB2: $\langle 489, NA \to ST \rangle$



(c) PB2: $\langle 489, ST \to NA \rangle$

Figure 3.10: The dominant support chains for mutations in H1N1 subtype. 1 means Years 2001-2003, 2 means Year 2002-2003, 3 means Years 2005-2007, 4 means Years 2007-2009, 5 means Years 1999-2001, 6 means Years 1976-1978

On H3N2 subtype

We also performed a similar analysis in H3N2, a human influenza virus. We detected 20 point mutations using our algorithm, as shown in Table 3.4 .

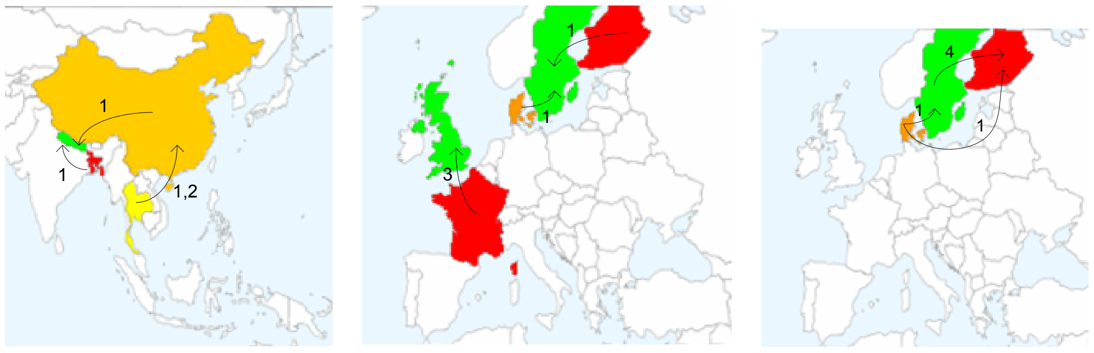
Again, one of the detected mutations, $\langle 176, K \to N \rangle$ (K176N for short) in HA protein turns out to cause the H3N2 outbreak in Nepal in 2004 [13]. Note that the position 176 in the aligned H3N2 HA protein corresponds to the position 145 in [13]. Figure 3.11(a) and 3.11(b) shows the spatiotemporal spread for K176N in Asia and Europe.

Table 3.4: The amino acid substitution in H3N2 subtype

Protein	Point Mutations
PB2	R340K,
PB1	I179M, K586R,
F2	Q25R, L82S
HA	D174N, **K176N**, **E188K**, G205D, V259I, D319G
NP	R77K, K98R, R103K, I197V, V197I
NA	N104D, G154V, E211K, I277T,

Another point mutation of interest is $\langle 188, E \rightarrow K\rangle$, which was reported to undergo multiple mutual substitutions [68]. Such spatiotemporal information are useful for the biologists to better understand the epidemiology of influenza, and in turn, to develop more effective vaccines (e.g. region-specific ones) to combat the spread of this fast changing virus.

Next, we focus on the mining of k-mutations and k-mutation chains on H3N2 subtype by allowing the deletion and insertion. We discover one valid pattern in HA protein: $\langle 390, IAGFIENGWEGM \rightarrow - - - - - - - - - - - -\rangle$. Its major spread chains are shown in Figure 3.11(c).



(a) HA: $\langle 176, K \rightarrow N\rangle$, Asia (b) HA: $\langle 176, K \rightarrow N\rangle$, Europe (c) HA: $\langle 390, IAGFIENGWEGM \rightarrow - - - - - - - - - - - -\rangle$

Figure 3.11: The dominant support chains for mutations in H3N2 subtype. 1 means Year 2003-2004, 2 means Year 2002-2004, 3 means Year 1992-1993, 4 means Year 2002-2003

3.5 Summary

The genetic structure of viruses is highly combinatorial in nature. Point mutations and gene segment exchange may occur anywhere along the primary sequence, contributing to a huge variability of viral protein products and the possibility of producing a new virus that can be easily transmitted between humans and initiate a pandemic. Effective influenza surveillance for pandemic preparedness is therefore critical to avoid the potentially deadly disaster for human kind. In particular, with the increasing availability of spatial and temporal information in the biological databases, new and advanced data analysis methods capable of rapid and in-depth genomic analysis that takes into account the spatiotemporal dynamics of the evolving viral species can help biologists to understand the evolution and circulation of the various viral species and to develop more effective and specific vaccines.

In this work, we have proposed a novel framework for discovering sequence mutations based on the mutation likelihood, including location and time of viral sequences and the sequence similarity. We designed an integrated algorithm to mine mutation chains in a top-down search manner and using two pruning strategies to reduce the search space. Experiments on synthetic datasets showed that our algorithm is more scalable and more efficient than the base line algorithms. Experiments on real world Influenza A virus database showed that our algorithms can discover meaningful mutations. Our methods are expected to provide an effective tool in the fight against emerging and re-emerging infectious diseases that are capable of rapid mutations and transmissions.

Chapter 4

Mining Global Interaction Pattern in Snapshot Data

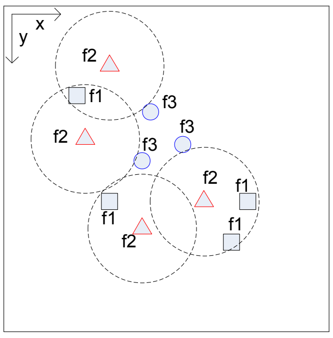
Besides the mutation patterns in biological sequences, another class of useful spatiotemporal patterns is localized and time-associated interaction patterns which are discovered in snapshot data. It is complex to mine localized and time-associated interaction patterns because both spatial point event types in snapshot data and spatiotemporal information are involved in the patterns. Therefore, we solve this mining problem in two steps: In this chapter, we first focus on the mining of interaction patterns among spatial point events on a single snapshot. In Chapter 5, we extend the work of this chapter to mine localized and time-associated interaction patterns.

Location-related patterns have many scientific applications [41, 63, 31, 90, 85]. For example, in epidemiology studies, dengue fever and Aedes mosquito tend to exhibit spatial correlation while in ecology, Nile crocodile and Egyptian plover are often found in tandem. Knowing the set of E-services that are located together is beneficial to mobile companies to improve their location-based services. The analysis of web log can also reveal the localized interests of customers in different geographical locations.

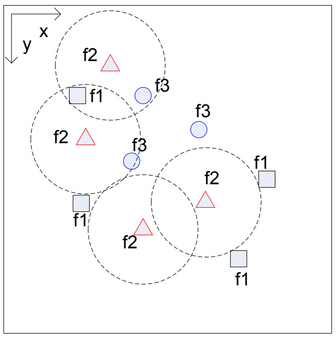
In view of this, there has been sustained interest in developing techniques to discover spatial collocation patterns [63, 31, 90, 85]. The interestingness measure of these patterns is a *binary* notion of proximity where a *distance threshold* is set and objects are either close or far away based on the threshold. Ripley's K function [58, 14] supports this measure by computing the probability of how objects of one feature is close to objects of one or more features. Interesting collocation patterns are then defined to be a set of features that are frequently close to each other [31, 90].

ID	X	Y	FID
001	22.8	27.5	f_1
002	32.2	60.8	f_1
003	75.1	60.4	f_1
004	70.3	72.5	f_1
005	32.4	17.7	f_2
006	25.1	40.2	f_2
007	42.4	67.0	f_2
008	61.0	61.2	f_2
009	45.2	33.0	f_3
010	54.9	37.0	f_3
011	40.1	48.2	f_3

(a) Dataset



(b) Observed Instance Distribution



(c) Underlying Instance Distribution

Figure 4.1: Some instances and their spatial relationship

Geo-spatial data are however by nature imprecise due to various reasons which include the limitation of measuring instruments, human recording errors, concern for privacy and dynamic movement of some objects. This means that no precise point can be used to represent the location of these objects.

Existing collocation mining algorithms, however, does not lend itself easily for handling uncertain spatial data. To address this problem, in this chapter, we will model the error distribution of the spatial data to be *Gaussian* instead of a precise point. Figure 4.1(a) gives a sample dataset of objects. Without special presentation, objects refer to the spatial instances in dataset in the rest of this chapter. With three feature types f_1, f_2, f_3, denoted by the symbols □, △, and ◯, respectively. Figure 4.1(b) gives the objects' distribution where the dotted circles define the object's distance threshold. We observe that many of the △ objects have □ objects within the distance threshold. In other words, {□, △} is a collocation pattern. However, if the exact locations of the □ instances are as shown in Figure 4.1(c), then {□, △} will not be a collocation pattern since many of their instance pairs are now outside the distance threshold. At the same time, many ◯ objects are now within the distance threshold of the △ objects.

The above example demonstrates that existing collocation mining approaches, which employ the exact support measure (i.e. the number of instances), are sensitive to the assigned distance threshold. As previously mentioned in Chapter 2, both snapshot-grid model and event model are sensitive to imprecise data. With different distance thresholds or noise in data, these algorithms may find different collocation patterns. In addition, they do not show good scalability as collocation mining procedure is, in essence, nothing more than the expensive spatial join among multiple datasets [90].

While one may build an uncertain model to capture the underlying distribution for each object, and derive the probability of an object being close to another object for a given distance threshold, such an approach is computationally expensive as each feature may have multiple objects.

Motivated by these challenges, we propose to model the spatial features in a continuous space using the radial basis functions. This approach resembles the kernel density estimation (KDE) [70] in statistics, but KDE does not consider the positional error of the uncertain data and focuses on the density estimation of only one feature.

In this chapter, we use two Gaussian functions, namely, the error function and the kernel function to model the observed position of the object. The actual influence is computed as the convolution of these two functions, which is still a Gaussian function with a wider bandwidth. By summing up the influences of all the instances of a feature, we obtain the influence distribution (or the influence map) of the feature.

We introduce the notion of *Spatial Interaction Patterns* (SIPs) to capture the interactions among sets of features. These patterns are sets of binary features whose influence maps are commonly correlated. For each feature type, we build an influence map that captures the distribution of the feature instances. Superimposing the influence maps allows the interaction of the feature types to be easily determined without costly spatial joins. Experiments are performed on both synthetic and real world datasets to demonstrate that the proposed approach is not only efficient but is able to discover patterns that have been missed by existing methods.

The remainder of this chapter is organized as follows. Section 4.1 proposes the influence model and its properties. Section 4.2 introduces the proposed mining algorithm called PROBER. We study the performance of the mining algorithm in Section 4.3 and summarize our finding in Section 4.4.

4.1 Influence Model

In this section, we introduce the notations used in defining the influence function to capture the degree of affinity between two spatial objects. We extend the notations to the influence maps of features, i.e., object groups, and infer some useful properties.

4.1.1 Object-to-Object Influence Function

Recall that most spatial data are inherently uncertain with an error distribution modeled by the Gaussian distribution $\mathcal{N}(\mu, \sigma^2)$,

$$e(\mu, x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{|\mu - x|^2}{2\sigma^2}}, \tag{4.1}$$

where μ is the observed i-th dimensional value and x is the underlying value.

When two spatial objects are near each other, they exert an influence on each other. This degree of influence is represented as a radial basis kernel function in the form of either Gaussian, Epanechnikov, Biweight or Triangle function. Let $k_i(\cdot)$ and $e_i(\cdot)$ denote the kernel function and error function, respectively, on the i-th dimension. Taking into account the effect of error distribution due to the uncertain spatial data, the actual influence exerted between two spatial objects along the i-th dimension is the convolution of $k_i(\cdot)$ with $e_i(\cdot)$, $k_i \otimes e_i$.

Definition 7. *The actual influence of an object on a point in the i-th dimension, denoted as inf_i, is defined as*

$$inf_i = k_i \otimes e_i \equiv \int_{-\infty}^{\infty} e_i(x)k_i(x-\tau)d\tau \tag{4.2}$$

where $k_i(\cdot)$ is the kernel function and $e_i(\cdot)$ is the error function, along the i-th dimension

.

In this work, we select the Gaussian function to be the kernel due to its unique influence range $(-\infty, \infty)$ among candidate kernels. Hence, the influence of an object on a point is the convolution of two Gaussian functions, namely Gaussian kernel and Gaussian error. We know that the convolution of two Gaussians is also a Gaussian function. Without loss of generality, let $e_i = \mathcal{N}(0, \sigma_e^2)$ and $k_i = \mathcal{N}(0, \sigma_k^2)$, $inf_i = k_i \otimes e_i$ $= \mathcal{N}(0, \sigma_e^2 + \sigma_k^2)$.

From this definition, we can easily generalize the influence function to the high dimensional space.

Definition 8. *Assuming that each dimension has identical Gaussian error $\mathcal{N}(0, \sigma_e^2)$ and Gaussian kernel $\mathcal{N}(0, \sigma_k^2)$. Given a d-dimensional object $o = (h_1, h_2, \dots, h_d)$, its* ***influence*** *to the neighboring point $p = (p_1, p_2, \dots, p_d)$ in the d-dimensional space, is the product of influence on each dimension:*

$$inf(o,p) = \prod_{i=1}^{d} \frac{1}{\sqrt{2\pi(\sigma_e^2 + \sigma_k^2)}} exp\{-\frac{(h_i - p_i)^2}{2(\sigma_e^2 + \sigma_k^2)}\} = (2\pi(\sigma_e^2 + \sigma_k^2))^{-d/2} exp\{-\frac{\sum_{i=1}^{d}(h_i - p_i)^2}{2(\sigma_e^2 + \sigma_k^2)}\} \tag{4.3}$$

In fact, the exponential factor $\sum_{i=1}^{d}(h_i - p_i)^2$ is the square of Euclidean distance of o and p in d-dimensional space, $(Euclidean(o, p))^2$. We observe that this influence function has the following properties. *Monotonicity*: It is anti-monotonic to the Euclidean distance between two objects in a high dimensional space; and *Robustness*: It takes into consideration the uncertainty in the data.

In the case of a 2D spatial object o on the x-y plane, its influence distribution is a bivariate Gaussian function whose mean is the observed position of o and standard deviation is $\sqrt{(\sigma_e^2 + \sigma_k^2)}$. If the kernel on each dimension has the same standard deviation (i.e., $\sigma_x = \sigma_y$), the influence distribution is circular in shape, otherwise, it is an ellipse. Figure 4.2 illustrate an influence function of circular shape. We call this bell-like 3D shape an **influence unit**. For the remainder of this chapter, we use $\mathcal{N}(0, \sigma^2)$ to denote an influence unit, where $\sigma = \sqrt{\sigma_e^2 + \sigma_k^2}$. Note that the circular region denotes a range of mean $\pm 3\sigma$ along the x-y plane and it captures over 95% of the influence exerted by the object.

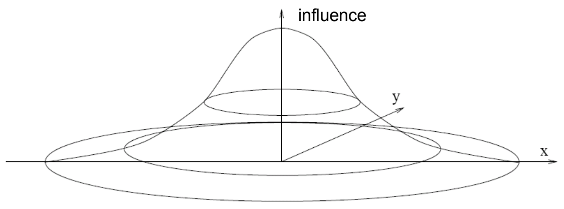


Figure 4.2: Influence distribution on 2D space

Lemma 5. *The influence measure is symmetric, i.e.,* $inf(o_i, o_j) = inf(o_j, o_i)$.

Proof: It can be inferred from the Definition 8 as the $Euclidean\ (o_i, o_j) = Euclidean(o_j, o_i)$.

□

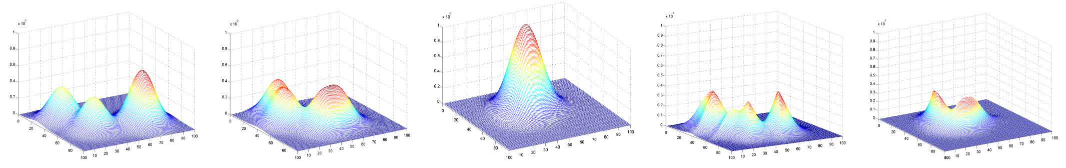

(a) $S(f_1)$ (b) $S(f_2)$ (c) $S(f_3)$ (d) $S(f_1, f_2)$ (e) $S(f_2, f_3)$

Figure 4.3: Examples of influence maps and their interaction

4.1.2 Feature-to-Feature Influence Function

Similar to the kernel density estimator [62], the influence map of a feature on a 2D plane is the normalized summation of all the instances' influence units of this feature. In our influence model, each object is assigned the identical bandwidth to model its influence. We do not focus on the selection of proper bandwidths to model influence maps, but smoothing techniques of kernel density estimator, which study the selection of proper bandwidths for different situations and are well-developed in statistics, can be easily adopted in our influence model.

Definition 9. *Given a set of spatial objects $\{o_1, o_2, \ldots, o_n\}$ of feature f on a spatial plane $\mathcal{P}$, and the influence function $inf(\cdot)$. The* **influence** *of feature f on a position $p \in \mathcal{P}$, denoted by $S(f, p)$, is*

$$S(f, p) = \frac{1}{n}\sum_{i=1}^{n} inf(o_i, p). \tag{4.4}$$

We use $S(f)$ to denote the **influence map** *of feature f.*

The volume of $S(f)$ can be computed as the integral of the influence of all points in $\mathcal{P}$. This leads to Lemma 6.

Lemma 6. *The volume of an influence map $S(f)$ is 1.*

Proof: Assume the feature f contains n objects $\{o_1, o_2, \ldots, o_n\}$.

Volume of $S(f) = \int_{p\in\mathcal{P}} S(f, p)dp$

$= \int_{p \in \mathcal{P}} \frac{1}{n} \sum_{i=1}^{n} inf(o_i, p) dp$

$= \frac{1}{n} \int_{p \in \mathcal{P}} inf(o_1, p) dp + \ldots + \int_{p \in P} inf(o_n, p) dp$

$= \frac{1}{n} \times n \times (Volume\ of\ influence\ unit)$

$= 1.$

$\square$

Definition 10. *Given two influence maps $S(f_1)$ and $S(f_2)$ with respect to feature f_1 and f_2 and the spatial plane $\mathcal{P}$, the* **influence of a feature pair** $\{f_1, f_2\}$ *on a position $p \in \mathcal{P}$, denoted as $S(f_1, f_2, p)$, is $min\{S(f_1, p), S(f_2, p)\}$. We use $S(f_1, f_2)$ to denote the* **influence map of the feature pair** $\{f_1, f_2\}$ *on plane $\mathcal{P}$.*

Definition 11. *The interaction between a pair of features $\{f_i, f_j\}$ is measured as the volume of the influence map $S(f_i, f_j)$. We call this measure the* **Interaction (I)** *of feature pair $\{f_i, f_j\}$ and is denoted as $I(f_i, f_j) = \int_{p \in P} S(f_i, f_j, p) dp$.*

The definition of influence map takes the minimum operator due to two reasons. First, it is consistent with all_confidence [52], a well-accepted correlation measure satisfying anti-monotone property: The interaction value decreases as the increase of features. Second, it is consistent with the definition of prevalence, a well-accepted measure of collocation patterns [63, 51, 31].

From Lemma 6, we infer that $0 \leq I \leq 1$. The interaction between a feature and itself is 1, i.e., $I(f_i, f_i) = 1$. Hence, $I(f_i, f_j) = 1$ indicates that the objects of feature f_i and f_j have the same distribution. On the other hand, $I(f_i, f_j) = 0$ implies that the data distributions of feature f_i and f_j are far apart from each other.

Figure 4.3 shows the influence maps for feature f_1 and f_3, and the feature pair $\{f_1, f_3\}$. Note that Definition 10 and Definition 11 can be easily extended to three or more features. For three features, we have $I(f_1, f_2, f_3) = \int_{p \in \mathcal{P}} min\{S(f_1, p), S(f_2, p), S(f_3, p)\} dp$. The Interaction measure (I) is used to determine the significance of a spatial interaction. This measure indicates how much a feature is affected by the interaction from the other features in a feature set. Lemma 7 and Lemma 8 gives some important properties of interaction measure.

Lemma 7. *(Symmetry property) The interaction measure is Symmetric, i.e.,* $I(f_i, f_j) = I(f_j, f_i)$.

Proof: It can be inferred from the Lemma 5 and Definition 11 as both *Euclidean*() and *min*() subfunctions are symmetric.

□

Lemma 8. *(Apriori property) The Interaction measure (I) is monotonically non-increasing as the increase of features.*

Proof: Let us assume that a feature set P_n consists of n features, $f_0, f_1, \ldots, f_{n-1}$, and the interaction of n features, $f_0, f_1, \ldots, f_{n-1}$, to be $I(f_0, f_1, \ldots, f_{n-1})$. According to Definition 10, $I(f_0, f_1, \ldots, f_{n-1}) = \int_{p\in\mathcal{P}} min\{S(f_0, p), S(f_1, p), \ldots, S(f_{n-1}, p)\}dp$. Then for a longer feature set $P_{n+1} = P \bigcup \{f_n\}$, we have

$I(f_0, f_1, \ldots, f_{n-1}, f_n)$

$= \int_{p\in\mathcal{P}} min\{S(f_0, p), S(f_1, p), \ldots, S(f_{n-1}, p), S(f_n, p)\}dp$

$= \int_{p\in\mathcal{P}} min\{min\{S(f_0, p), \ldots, S(f_{n-1}, p)\}, S(f_n, p)\}dp$

$\leq \int_{p\in P} min\{S(f_0, p), S(f_1, p), \ldots, S(f_{n-1}, p)\}\}dp$

$= I(f_0, f_1, \ldots, f_{n-1})$.

□

Lemma 7 implies that we can construct an undirected graph where each node indicates a feature and the edges associated with node pairs indicate the interaction. Lemma 8 implies that the pattern generation may follow the classic Apriori property, avoiding some patterns which are impossible to be valid.

4.2 Mining Spatial Interaction Patterns

In this section, we present the algorithm PROBER (sPatial inteRactiOn Based pattErns mineR) to find the interaction patterns in spatial databases.

Definition 12. *Given a spatial database containing a feature set $\mathcal{F}$ and a threshold min_I, a* **Spatial Interaction Pattern** *SIP is the set of features $\{f_1, f_2, \ldots, f_k\} \subseteq \mathcal{F}$ and $I(f_1, \ldots, f_k) \geq min\_I$.*

If the interaction of a SIP is greater than a predefined threshold min_I, we call this SIP to be a *valid* SIP or *frequent* SIP. A SIP P_1 is a *subpattern* of another SIP P_2 if $P_1 \subseteq P_2$. For this case, we also say P_2 is a *superpattern* of P_1. Due to the Apriori property, if a SIP is valid, any one of its subpatterns is a valid SIP. We say that a valid SIP *covers* its all subpatterns.

A SIP P is a *maximal* SIP if 1) P is a valid SIP, and 2) there does not exist any its superpattern P' such that P' is a valid SIP. The *maximal* SIP set is the set of maximal SIPs. For example, given the valid SIPs $\{\{f_1, f_2\}, \{f_2, f_3\}, \{f_1, f_3\}, \{f_3, f_4\}, \{f_1, f_2, f_3\}\}$, the maximal SIP set is $\{\{f_3, f_4\}, \{f_1, f_2, f_3\}\}$.

The problem of mining spatial interaction patterns is defined as follows. *Given a spatial database containing m features and n instances, as well as the minimal interaction measure min_I, our goal is to find the maximal SIP set.*

Mining of SIPs is computationally expensive due to two-fold reasons. First, the comparison of continuous spaces is computationally infinite. Second, the enumeration of all candidate patterns is exponential. We consider the first problem in Section 4.2.1 and examine the second problem in Section 4.2.2 and 4.2.3. In Section 4.2.4, we present the PROBER algorithm and analyze its complexity.

4.2.1 Uniform Sampling Approximation

In theory, the influence map of one feature is continuous, which means that there are infinite comparisons when considering the relationship between two influence maps. To expedite the mining process, we uniformly divide the spatial plane into disjoint cells and only the centers of these cells serve as the positions of influence. In this way, the comparison between influence maps is reduced to cell comparisons.

We use *progressive refinement approach* to build approximate influence maps by allowing errors. Assume that the target geographical plane is a square of length L. Given a resolution R, we divide the plane into $\frac{L}{R} \times \frac{L}{R}$ cells. For each cell, we use the center of the cell to approximate the influences exerted on this cell by other objects in the neighbouring cells. The parameter R determine the resolution of this approximation. As long as R is sufficiently small, our model will provide a good approximation. We denote this the approximation of S to be $\widehat{S}$. One issue is to estimate the upper of this approximation error. We define the **Influence Error (IErr)** as follows.

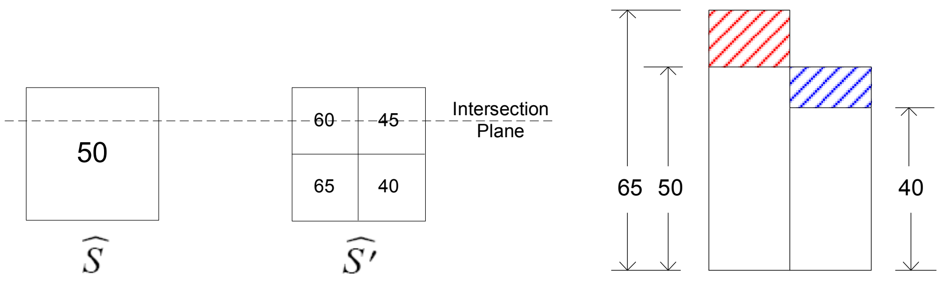
Definition 13. *Suppose we represent the approximate influence map* $\widehat{S}$ *as a* $[n \times m]$ *matrix. For any granularity coefficient, c, the refined approximate influence map* $\widehat{S'}$ *is a* $[c \cdot n \times c \cdot m]$ *matrix. The difference in the two influence maps is given by:*

$$IErr(\widehat{S'}, \widehat{S}) = \frac{1}{|\widehat{S'}|} \times \sum_{every\ cell \in \widehat{S'}} \frac{\|\ cell_{\widehat{S'}} - cell_{\widehat{S}}\ \|}{MAX(cell_{\widehat{S'}}, cell_{\widehat{S}})} \tag{4.5}$$

where $|\widehat{S'}|$ *denotes the matrix size of* $\widehat{S'}$*,* $cell_{\widehat{S'}}$ *is a single cell at* $\widehat{S'}$*, and* $cell_{\widehat{S}}$ *is the cell which covers* $cell_{\widehat{S'}}$ *at matrix* $\widehat{S}$*.*

For example, Figure 4.4 shows $\widehat{S}$ and $\widehat{S'}$, respectively, and one view of the intersection plane. In this example, $IErr$ is the volume of the shaded areas in the right part of Figure 4.4. $IErr = \frac{1}{4} \times (\frac{|60-50|}{60} + \frac{|65-50|}{65} + \frac{|45-50|}{50} + \frac{|40-50|}{50}) \approx \frac{1}{4} \times 0.69 = 0.17$.

Note that although the term ”resolution” used here is similar to the bin width in his-



(a) Influence Maps (b) Intersection

Figure 4.4: An example to compute influence error

togram theory in statistics [62], our approximation approach is different. In statistics, a fine bin width is selected to avoid under-smoothing while estimating data distribution [70], whereas our goal is to compare multiple influence maps (estimators in KDE) efficiently within an acceptable error tolerance. In this sense, bin width selection in histogram theory is not suitable for our mining requirement. Hence, we design a new mechanism to select the proper resolution for a given error bound.

Lemma 9. *As $\theta = \frac{R}{\sigma}$, the error bound $IErr$ between the approximate influence map of resolution r, $\widehat{S|_{R=r}}$, and the space of resolution $\frac{r}{2}$, $\widehat{S|_{R=r/2}}$, is*

$$IErr(\widehat{S|_{R=r}}, \widehat{S|_{R=r/2}}) \leq \frac{1 - e^{-\frac{R^2}{16\sigma^2}} e^{-\frac{kR}{4\sigma}} + e^{\frac{kR}{4\sigma}} - e^{\frac{R^2}{16\sigma^2}}}{2}, \tag{4.6}$$

where $0 \leq k \leq 3$.

Proof: Appendix gives the proof for Formula 7.6. Here, we extend the proof to Formula 4.6. This is done in two steps. First, for one particular cell and a set of objects, Formula 7.6 holds because the influence from a set of n objects to the center position p of one cell is $\frac{1}{n}\sum_{i=1}^{n} inf(o_i, p)$ given by Definition 9. Next, for all cells on the plane and the set of objects, the influence error is essentially the normalized combination of every singular cell given by Definition 13, so Formula 7.6 still holds. Hence, we conclude $\widehat{S(\cdot)|_{R=r}} = \sum_{i=1}^{n} \sum_{all\ cells} inf(o_i, cell\ center)$.

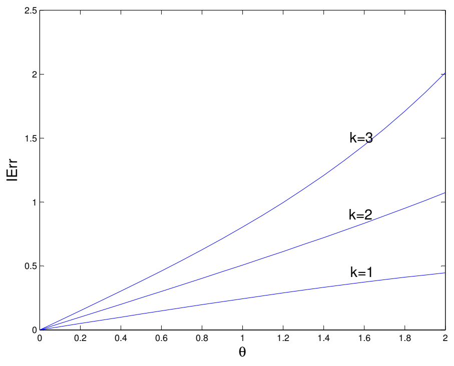


Figure 4.5: The error bound of influence error

□

Figure 4.5 shows the error bounds as we vary k, where the x- coordinate is the ratio $\theta = \frac{R}{\sigma}$, y- is the possible fluctuation over the real influence. The $k = 3$ curve is the *error bound* when $0 \leq k \leq 3$. The worst case error occurs when a split is performed on the marginal cells of an influence unit. In practice, this does not happen often. As a result, the influence fluctuation is far less than the error bound. The real error curve will depends on the data distribution, which is supported by experiments in Section 4.3.1. In fact, we have the Theorem 2 no matter the data distributions are.

Theorem 2. *As $\theta = \frac{R}{\sigma} \to 0$, the approximate influence map $\widehat{S}$ converges to the real influence map S.*

Proof: With an initial resolution r, we obtain the approximation space $\widehat{S|_{R=r}}$. Next, we halve the resolution to obtain its finer approximation space $\widehat{S|_{R=r/2}}$. Lemma 9 gives the upper bound and lower bound of the ratio of the two approximation spaces. By iterating this operation k steps, we have the approximation spaces of $\theta = \frac{R}{\sigma} = \frac{r}{2^k \times \sigma} \to 0$. From Formula 4.6 and Figure 4.5, we have $\frac{\widehat{S}}{S} \to 1$, which completes the proof.

□

Algorithm. Let $O = \{o_1, o_2, \ldots, o_n\}$ be a set of objects of one particular feature. To obtain the approximate influence map of this feature, we employ the BuildApproSpace algorithm.

Algorithm 5: BuildApproSpace

```
input  : Dataset $O$;
         kernel deviation $\sigma$;
         inital resolution $r$;
         error bound $min\_err$.
output: Approximate influence map $\widehat{S(\cdot)}$.
1 Initialize two approximation spaces $S_b = \widehat{S(\cdot)|_r}$ and $S_f = \widehat{S(\cdot)|_{r/2}}$;
2 while $(IErr(S_f, S_b) > min\_err)$ do
3     $S_b \leftarrow S_f$;
4     $r \leftarrow \frac{r}{2}$;
5     Initialize $S_f = \widehat{S(\cdot)|_{r/2}}$;
6 return $S_f$.
```

Line 1 of Algorithm 5 builds two spaces, S_f and S_b. S_f is implemented as a $\frac{L}{r} \times \frac{L}{r}$ matrix while S_b is a $\frac{2L}{r} \times \frac{2L}{r}$ matrix, where L is the plane width. For each object $o_i \in O$, we superimpose a minimal bounding rectangle (MBR) of side 6σ onto the two spaces, centering at the position of o_i. This results in the updates of element values on the two spaces respectively.

To compare the two matrices S_f and S_b, for each element of S_b, we find the corresponding four elements in matrix S_f and obtain the absolute difference among them. Line 2 computes the approximate error within each cell of S_b individually, and take the arithmetic average which is given by Definition 13.

If the approximation error is greater than the user specified parameter min_err, we initialize a new matrix at half the resolution. We compute the approximation error of this finer resolution space. This process repeats until the error is less than min_err, and S_f is the final approximate influence map, which is guaranteed by Theorem 2.

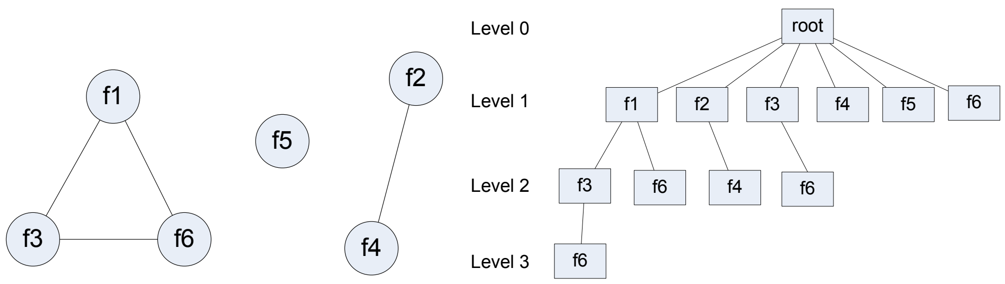
4.2.2 Pattern Growth and Pruning

Lemma 8 indicates that the interaction measure satisfies the downward closure property. In other words, a candidate pattern is possible to be valid only if all its subpatterns are valid. This allows many SIPs to be pruned during the mining process.

However, the number of valid SIPs can be very large, since a valid SIP of n features has $(2^n - 1)$ subsets that are also valid SIPs. The majority of these valid SIPs are redundant as their interaction can be inferred from their superset. Avoiding the generation of these redundant SIPs can significantly improve mining efficiency and save memory space. Current state-of-the-art maximal pattern mining algorithms [26, 59] use a search tree structure to facilitate depth-first search to find frequent itemsets, but they cannot deal with maximal interaction pattern mining problem directly.

Motivated by the idea of maximal pattern mining algorithms, we employ a depth-first search with "look ahead". A tree structure called *interaction tree* is used to facilitate the mining process. It is similar to the search tree in [26, 59], but with one important extension. Each node at level 2 denotes an interaction pattern of 2 features, and its associated influence map. Algorithm 6 gives the details.

Assuming that we have obtained all the valid interaction patterns of size 2, denote by C_2. In order to visualize the relationships among feature sets, an *interaction graph* can be constructed beforehand, in which each feature is a node in the graph, and two nodes are connected by one edge if they are correlated (or exist in C_2). Obviously, the interaction graph is an undirected graph due to the symmetric property given in Lemma 7. For example, the correlated pairs from the database forms an interaction graph in Figure 4.6(a). There are four edges indicating four pairs of correlated features.



(a) An interaction graph (b) An interaction tree

Figure 4.6: Data Structure for Mining Maximal SIPs

We further transform the interaction graph to an interaction tree as follows. A root node is first created at level 0. At level 1, we create a node for each feature as a child of the root, and the order of children follows sthe lexigraphical order. In the subsequent levels, for each node u at level k ($k > 1$) and for each right sibling v of u, if $((u, v))$ is connected in the interaction graph (namely $\{u, v\}$ is an interaction pattern), we create a child node for u with the same label of v. For each node, we could enumerate one candidate pattern, with prefix feature set of its parent node by concatenating the feature in this node. For example, we construct the tree shown in Figure 4.6(b) based on Figure 4.6(a).

Note that if a pattern p is not a valid SIP, then any longer pattern that contains p cannot be a valid SIP. This allows us to effectively prune off unnecessary computations. Further, the structure of the interaction tree always forces the evaluation of the longest patterns first. This implies that if the longest pattern is a valid SIP, then we do not need to evaluate any of its sub-patterns.

4.2.3 Interaction Tree Traversal

The evaluation of SIP is performed by Algorithm 6. Given a feature node f_n in the interaction tree, Line 1 obtains the parent node of f_n. Line 2 forms a candidate pattern P_{cand} by backtracking from the current node to root of interaction tree. As an example, for the feature node f_6 at level 3 in Figure 4.6(b), we can backtrack f_6 to form a candidate pattern $\{f_1, f_3, f_6\}$ with prefix $\{f_1, f_3\}$.

Since the computation of influence map interaction is expensive, we postpone this computation until it becomes necessary. As long as there is one superset of P_{cand} in C, this computation can be delayed. Line 4 sets this node to be a *delay* node and Line 5 propagates this to its children nodes. Lines 7-13 recover the influence map of the prefix. For example, suppose we already have a maximal SIP $\{f_1, f_2, f_4, f_5\}$ in maximal SIP set

C. The candidate patterns $\{f_1, f_4\}$ and $\{f_1, f_4, f_5\}$ can be exempt from evaluating as they are both subpatterns of $\{f_1, f_2, f_4, f_5\} \in C$. Here f_4 and f_5 are marked *delay* nodes in interaction trees. If we need to evaluate another candidate pattern $\{f_1, f_4, f_5, f_6\}$, the interaction computation of its prefixes $\{f_1, f_4\}$ and $\{f_1, f_4, f_5\}$ becomes necessary. So the influence map computation will start from f_4 via f_5 along the path to f_6. The pseudocode of this operation is given in Lines 7-13. Finally, Line 13 obtains the final influence map of prefix, $parS$.

Lines 14-18 compute the interaction between the influence maps of the prefix node and the current node in a depth-first manner. If the interaction is no less than the threshold *min_I*, this candidate pattern is valid, and it will be added to the maximal pattern set C in Line 16.

Although algorithm $EvalSIP(\cdot)$ is devised to find maximal SIPs, it is capable to discover all valid SIPs. The idea is straightforward: the subpatterns checking in maximal SIP set C is skipped, which is achieved by deleting Lines 3-11 in Algorithm 6.

4.2.4 Algorithm PROBER

We now present the Algorithm PROBER to mine spatial interaction patterns. The algorithm incorporates the pattern enumeration technique into the mining process. Algorithm PROBER takes as input the spatial database D, the influence error threshold *min_err*, the interaction measure threshold *min_I*, and outputs the set of maximal SIPs.

Line 1 finds the feature set from the dataset. Lines 3-4 build the approximate influence maps for each feature, by calling the $BuildApproSpace(\cdot)$ algorithm. To facilitate the next mining phase, the approximate influence maps of all features are required to be superimposed using the same resolution. Therefore, the halt condition in $BuildApproSpace(\cdot)$ is modified to be "If the maximal $IErr(S_b, S_f)$ of all features is greater than *min_err*, then do the next iteration".

Algorithm 6: $EvalSIP(f_n, min\_I, C)$

```
1  $parNode \leftarrow$ the parent node of $f_n$;
2  backtracking $fn$ till root to form a candidate $P_{cand}$;
3  if $P_{cand}$ has a superpattern in $C$ then
4      set $f_n$ to be a $delay$ node;
5      $EvalSIP(fn's\ child\ node, min\_I)$;
6  else
7      $parS$ = a unit matrix;
8      while $parNode$ is a delay node do
9          $parS = Interaction(parS, S(parNode))$;
10         $parNode \leftarrow$ the parent node of $parNode$;
11     $parS2 \leftarrow$ the influence map of $parNode$;
12     $parS = Interaction(parS, parS2)$;
13     $fS = Interaction(parS, S(f_n))$;
14     if $fS > min\_I$ then
15         add $P_{cand}$ to $C$;
16         call $EvalSIP(f_n's\ child\ node, min\_I, C)$;
17     else
18         call $EvalSIP(f_n's\ sibling\ node, min\_I, C)$;
```

Lines 6-10 discover the interaction in all feature pairs combination. In particular, Line 9 computes the interaction of the one feature pair by taking the minimal value between each element pair and summing up all the minimal values. If the measure is greater than *min_I*, this feature pair is considered to be correlated.

Line 11 builds the interaction tree using the set of interaction pairs as the tree proposed in [38]. Line 13 invokes algorithm $EvalSIP(\cdot)$ to recursively visit the necessary feature nodes in interaction tree, starting from the root node of tree. Finally, Line 14 returns the maximal pattern set C.

Continuing with our example in Figure 4.1, we assume σ=10 and *min_I*= 0.3, Table 4.1 shows the mining process of PROBER. The mining stops at level 2 because the pattern $\{f_1, f_3\}$ is not a valid SIP, hence the pattern $\{f_1, f_2, f_3\}$ is pruned. As a result, the maximal SIPs are $\{\{f_1, f_2\}$ and $\{f_2, f_3\}\}$.

Complexity Analysis. Let $\theta=\sigma/R$ where R is final resolution after multiple itera-

Algorithm 7: PROBER

```
   input : $D$: the spatial database;
           $\sigma$: kernel deviation;
           $min\_err$: influence error threshold;
           $min\_I$: interaction threshold.
   output: $C$: the set of SIPs.
1  Let $RF$ to be all features in $D$;
2  /*Phase 1: impose approximate influence map*/ ;
3  for each feature $f_i \in RF$ do
4      call $BuildApproSpace(\cdot)$ to build the influence map;
5  /*Phase 2: build interaction tree*/ ;
6  $C = \emptyset$;
7  Impose an ordering on $RF$;
8  for each feature pair $(f_i, f_j)$, where $f_i \prec f_j$ do
9      evaluate feature pair $(f_i, f_j)$;
10     if $I(f_i, f_j) \geq min\_I$, add pattern $\{f_i, f_j\}$ to $C$;
11 build the interaction tree $T_{col}$ based on $C$;
12 /*Phase 3: mine maximal SIP*/;
13 call $EvalSIP(T_{col}.root, min\_I, C)$;
14 return $C$.
```

tions. To build the influence map (i.e. $BuildApproSpace(\cdot)$ algorithm), an influence unit of range $6\sigma \times 6\sigma$ is computed for each instance in the database, thus it needs $(6\sigma/R)^2$=$(6/\theta)^2$ distance computation. The overall computational complexity to build the influence maps is $O(n(6/\theta)^2)$, where n is the database size. Since there is one influence map matrix for each feature, the space complexity is $O(f(\frac{L}{R})^2)$ where L is the plane length, f is the feature number.

The PROBER algorithm, in the worst case, could generate $\binom{f}{\lfloor \frac{f}{2} \rfloor}$ maximal SIPs, and each of them requires $\lfloor \frac{f}{2} \rfloor$ influence map comparison along the path from root to the leaf node of the maximal SIP. Each influence map comparison requires the computation of complexity $O((\frac{L}{R})^2)$. Hence, the overall computational complexity in mining phase, is $O(\binom{f}{\lfloor \frac{f}{2} \rfloor} \times \lfloor \frac{f}{2} \rfloor \times (\frac{L}{R})^2)$. The space complexity include the space to store interaction tree and influence matrixes. The interaction tree has maximal 2^f nodes, and each node store an influence matrix. As the space required for influence matrix dominates, the worst

Table 4.1: Mining SIPs by influence model

level=1		
f_1	$S(f_1)$ (see Figure 4.3(a))	$I(f_1)$=1
f_2	$S(f_2)$ (see Figure 4.3(b))	$I(f_2)$=1
f_3	$S(f_3)$ (see Figure 4.3(c))	$I(f_3)$=1
level=2		
$\{f_1, f_2\}$	$S(f_1, f_2)$ (see Figure 4.3(d))	$I(f_1, f_2)$=0.58
$\{f_1, f_3\}$	$S(f_1, f_3)$	$I(f_1, f_3)$=0.27
$\{f_2, f_3\}$	$S(f_2, f_3)$ (see Figure 4.3(e))	$I(f_2, f_3)$=0.38

space requirement is $O((2^f - f) \times (\frac{L}{R})^2)$.

In summary, the computational complexity of PROBER is $O(n(6/\theta)^2 + \binom{f}{\lfloor \frac{f}{2} \rfloor} \times \lfloor \frac{f}{2} \rfloor \times (\frac{L}{R})^2)$, and the space complexity of PROBER is $O(2^f(\frac{L}{R})^2)$.

4.3 Experimental Studies

In this section, we present the results of experiments to evaluate the performance of PROBER. We also compare PROBER with existing collocation algorithms FastMiner [90] and TPMiner [77]. In order to set reasonable comparison, we generate two versions of PROBER, PROBER_ALL to discover all patterns, and PROBER_MAX to discover only maximal patterns. Note that while comparing the effectiveness of PROBER with FastMiner and TPMiner may not be appropriate due to the different interesting measures used, however, we could treat FastMiner and TPMiner as good baselines w.r.t both effectiveness and scalability issues. Table 4.2 show the parameter counterparts between influence model and distance model. In the following experiments, we assign the identical values to the parameter counterparts, e.g. $\sigma = d = 50$ and $min\_I = min\_prev = 0.4$.

Synthetic Datasets: We extend the synthetic data generator in [77] to generate the synthetic spatial databases with Gaussian noise. All the data are distributed on the plane of 8192×8192. The synthetic datasets are named using the format "Data-(m)-(n)-(d)-

Table 4.2: Parameter counterparts

Distance Model	Influence Model
distance threshold: d	influence deviation: σ
minimal prevalence: min_prev	minimal interaction: min_I

(N)” where m is the confidence feature number, n is the non-confidence feature number, d is the distance threshold and N is the number of instances in the dataset. For example, DATA-8-2-50-200k is a dataset which contains 8 confident features, 2 noise features, distance threshold to be 50, and a total of 200k instances. For each object, we assign a Gaussian noise of 0 mean and σ_e, say 5, deviation on each dimension.

Real-life Datasets: The real-life dataset used in our experiments is the *DCW environmental data.* We downloaded 8 layers of Minnesota state from Digital Chart of the World [1] (DCW). Each layer is regarded as a feature in our experiments as shown in Table 4.5. We further map the dataset on a formal 8192×8192 2D plane.

All the algorithms are implemented in C++. The experiments were carried out on a Pentium 4 3Ghz PC with 1GB of memory, running Windows XP.

4.3.1 Performance of Influence Map Approximation

In this section, we evaluate the convergence and efficiency of the $BuildApproSpace$ algorithm on both synthetic data Data-6-2-100-50k and DCW data. We assign the initial resolution $r = 256$ and $min\_err = 0.05$, as well as $\sigma = 100$ for Data-6-2-100-50k and $\sigma = 50$ for DCW data. Table 4.3 and 4.4 shows the results. In both tables, each row gives the influence error $IErr$ for each iteration. We can see that the $IErr$ converges to zero with each iteration.

The time taken for each iteration is shown in Figure 4.7. We observe that the runtime increases quadratically as the number of iteration increases, as the resolution and θ are

[1]http://www.maproom.psu.edu/dcw

Table 4.3: Convergence on DCW data

Iteration	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	MAX
1	0.16	0.02	0.15	0.13	0.16	0.04	0.17	0.17	0.17
2	0.10	0.01	0.09	0.08	0.10	0.03	0.11	0.11	0.11
3	0.06	0.01	0.05	0.05	0.06	0.02	0.06	0.07	0.07
4	0.03	0.00	0.03	0.03	0.03	0.01	0.04	0.04	0.04

Table 4.4: Convergence on Data-6-2-100-50k

Iteration	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	MAX
1	0.17	0.17	0.17	0.17	0.17	0.17	0.25	0.24	0.25
2	0.10	0.10	0.10	0.10	0.10	0.10	0.15	0.14	0.15
3	0.05	0.06	0.06	0.06	0.06	0.06	0.08	0.08	0.08
4	0.03	0.03	0.03	0.03	0.03	0.03	0.04	0.04	0.04

decreased by half. This is expected because a finer resolution and smaller θ will cause a quadratic increase in both time and space complexity. On the other hand, the runtime is linear to the database size for each iteration, as the DCW data contains 2837 objects as a whole and the synthetic dataset contains 50k objects. The results in Figure 4.7 are consistent with the analysis in Section 4.2.4.

4.3.2 Effectiveness Study

In this set of experiments, we show that PROBER_ALL algorithm is more robust than FastMiner and TopologyMiner as the variation of deviation/distance threshold. As the results of FastMiner and TopologyMiner are exactly same, we only compare PROBER_ALL with TopologyMiner in our experiment.

We first use a synthetic dataset Data-5-0-50-5k to evaluate effectiveness. Without noise, the expected patterns will be the maximal pattern of 5 confident features and all its sub-patterns, i.e. 26 patterns (2^5-5-1=26). We integrate different Gaussian noise of

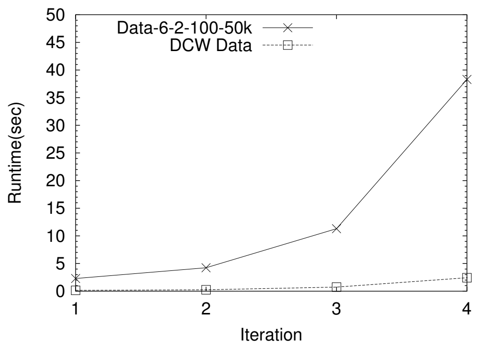


Figure 4.7: Convergence Performance

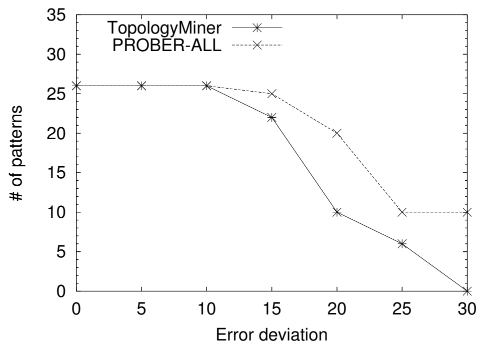


Figure 4.8: Effectiveness study

Table 4.5: Feature Description

FID	Name	Number of Points
f_0	Populated Place	517
f_1	Drainage	6
f_2	Drainage Supplemental	1338
f_3	Hypsography	72
f_4	Hypsography Supplemental	687
f_5	Land Cover	28
f_6	Aeronautical	86
f_7	Cultural Landmarks	103

identical mean 0 but different deviation ranging from 1 to 30. The comparison measure is the number of interaction patterns discovered, including maximal and non-maximal ones. The results are shown in Figure 4.8. From this figure, we observe that both PROBER_ALL and TopologyMiner can find the whole set of possible patterns while error deviation is less than 10. This is expected because the small error deviations do not have impact on the influence deviation. Therefore TopologyMiner can find all of the patterns with noise of deviation 10. On the other hand, the patterns found by TopologyMiner show greater decrease than the ones by PROBER_ALL, as the increase of error deviation.

Next, we apply PROBER_ALL on the DCW environment data. We set the interaction threshold *min_I* to be 0.4. The mining results are shown in Table 4.6. We observe that regardless of how the σ varies, the patterns discovered by PROBER_ALL

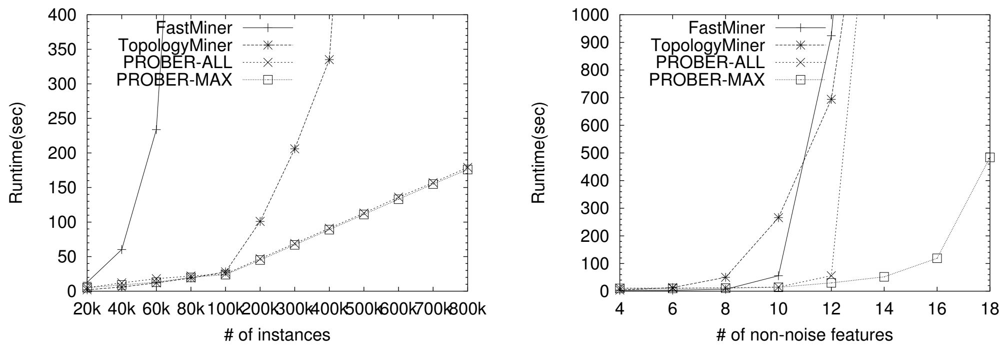
Table 4.6: Patterns Comparison of DCW Dataset

d/σ	Distance Model	Influence Model
50	NA	$\{f_0, f_2\}$, $\{f_2, f_4\}$
100	$\{f_0, f_2\}$, $\{f_2, f_4\}$	$\{f_0, f_2\}$, $\{f_0, f_4\}$, $\{f_0, f_6\}$, $\{f_2, f_4\}$
150	$\{f_0, f_2\}$, $\{f_0, f_4\}$, $\{f_2, f_4\}$	$\{f_0, f_2\}$, $\{f_0, f_3\}$,$\{f_0, f_4\}$, $\{f_0, f_6\}$,$\{f_0, f_7\}$, $\{f_2, f_3\}$, $\{f_2, f_4\}$, $\{f_2, f_6\}$, $\{f_3, f_4\}$,$\{f_3, f_7\}$, $\{f_4, f_6\}$,$\{f_4, f_7\}$, $\{f_6, f_7\}$
200	$\{f_0, f_2, f_4\}$, $\{f_3, f_4\}$	$\{f_0, f_2, f_4\}$, $\{f_0, f_2, f_6\}$, $\{f_0, f_6, f_7\}$, $\{f_0, f_3\}$, $\{f_2, f_3\}$, $\{f_2, f_7\}$, $\{f_3, f_4\}$,$\{f_3, f_6\}$,$\{f_3, f_7\}$, $\{f_4, f_6\}$,$\{f_4, f_7\}$

are always a superset of those found by the two distance model-based techniques, FastMiner and TopologyMiner. In particular, when $\sigma = 200$, we find that {populated place, drainage supplemental, hypsography supplemental} and {populated place, aeronautical, cultural landmarks} are missed by the distance model-based techniques but are discovered as SIPs.

4.3.3 Scalability

In this set of experiments, we demonstrate the scalability of both PROBER_ALL and PROBER_MAX. We set the number of features to 10 (including non-noise and noise features), and generate twelve datasets Data-8-2-50-{20k, 40k, 60k, 80k, 100k, 200k, ..., 800k}. We compare the performance of PROBER with FastMiner and TopologyMiner by varying the total number of instances. Figure 4.9(a) shows that both FastMiner and TopologyMiner increase exponentially as the number of instances increases while PROBER shows a linear increase. This is expected because the time complexity for the distance model is polynomial time of the number of instances while PROBER_ALL and PROBER_MAX are linear to the number of instances during the database scan and is independent of the database size during mining phase. In further observation, PROBER_MAX is slightly faster than PROBER_ALL because PROBER_MAX



(a) Effect of DB size (b) Effect of non-noise feature number

Figure 4.9: Scalability study

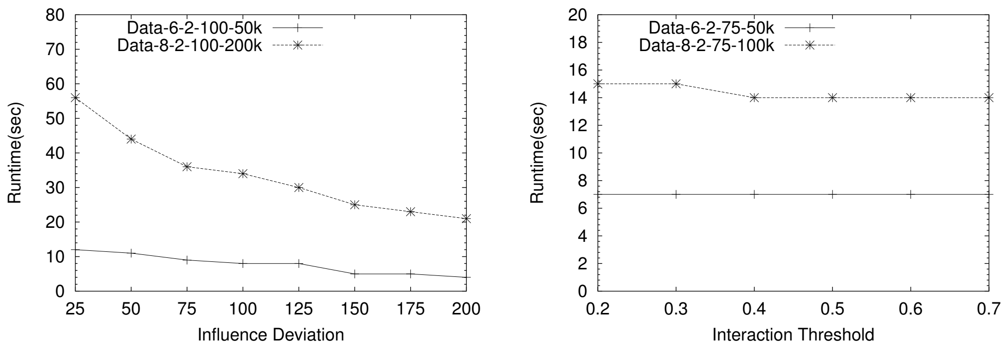
only detects the maximal patterns which can save the mining cost.

We also set the database size at 20k instances and generate eight datasets Data-{4, 6, 8, 10, 12, 14, 16, 18}-0-50-20k to evaluate the three algorithms. The results are shown in Figure 4.9(b). Both FastMiner and TopologyMiner do not scale well w.r.t the number of non-noise features. TopologyMiner allows pattern growth in a depth-first manner, but the extraction of project databases requires much time and space. PROBER_MAX shows the best scalability compared to the other algorithms, although the algorithm slows down when the number of features exceeds 16. This is because of the large confidence features results in the exponential growth of its interaction tree.

4.3.4 Sensitivity

Finally, we examine the effect of two parameters, influence deviation σ and interaction threshold *min_I*, on the performance of PROBER. Due to the intrinsic difference between influence model and distance model, it is unfair to compare the sensitivity performance of the two models. Therefore, we only include the influence model in this experiment.

Effect of Influence Deviation (σ). We first evaluate the effect of the influence deviation on PROBER_MAX. The two datasets used in this experiment are Data-6-



(a) Effect of influence deviation (b) Effect of interaction

Figure 4.10: Sensitivity study

2-100-50k and Data-8-2-100-200k, which imply that the patterns will be valid once the σ surpass 100. Figures 4.10(a) gives the results. PROBER_MAX run faster as the increase of σ. This is expected due to two reasons: 1) The time complexity of PROBER are inversely related to the σ, consistent to the complexity analysis in Section 4.2.4; 2) Bigger σ implies smoother distribution of influence maps, so it incurs less iteration rounds to build approximate spaces with error $IErr$, which saves the cost. On the contrary, the algorithms of distance model are sensitive to distance threshold due to the tremendous increase of time cost [77].

Effect of Interaction Threshold (*min_I*). We evaluate the three algorithms on two dataset Data-3-3-50-20k and Data-5-5-50-50k with potential prevalence is 0.5, which implies that the PROBER may find many patterns while $min\_I < 0.5$. From Figures 4.10(b), PROBER_MAX are not sensitive to min_I because the mining cost is not the dominant factor compared with the cost to build approximate spaces. On the contrary, the runtime of FastMiner and TopologyMiner will decrease as prevalence increases. This is expected because less patterns become frequent as the increase of prevalence, which leads to reduced mining cost for the two algorithms.

4.4 Summary

In this chapter, we introduce an influence model to present the spatial distribution of event data and analyze the bounds of computational error for building influence maps. Compared to the distance model used in existing works, the influence model considers the spatial affinity in terms of continuous functions instead of discrete functions. This leads to more meaningful mining results. Another advantage of the influence model is that it avoids expensive join operations, which are traditionally required to discover the relationship among spatial instances. We also introduce the concept of Spatial Interaction Patterns (SIPs) and design an approximate mining algorithm PROBER using influence model to find maximal SIPs. The experiment results on both synthetic and real-life datasets demonstrate that PROBER is effective and scalable.

Chapter 5

Mining Interaction Pattern Chains in Snapshot Data

In the previous chapter, we address the problem of mining global interaction patterns on a single snapshot, where the interaction patterns hold on the *whole* spatial plane of the snapshot. In this chapter, we focus on the problem of mining localized and time-associated interaction patterns, which are the patterns supported by the confined regions in some consecutive snapshots.

To find localized and time-associated interaction patterns are important to satisfy the application requirement of Location-Based Services (LBS). LBS are applications that take the geographical-related information into account and focus on the local data analysis and the local knowledge. Though the term of LBS has traditionally been used to refer to mobile device services using the Global Positioning System (GPS), in the recent years, it has been extended to web applications since the web resources contain a plenty of location information. The location information of web resources include three categories [76] as follows.

- Provider location: The physical location of the provider who owns the web service, such as organization, corporation or person. This kind of location is crucial to web geographical information retrieval and navigation such as online map and Yellow Pages services.

- Content location: The geographical location where the web resources describe, such as the location names or the geographical-related names. The content locations are utilized to improve the performance of information retrieval based on the location of user.
- Serving location: The geographical scope that a web resource can reach. For example, if a web resource is visited by the people in Singapore, its serving scope is Singapore. Knowing the serving location of a web resource can benefit many business applications such as local advertisements and e-commerce.

Many application and research efforts have been made in content locations. The commercial search engines, including Yahoo! Local and Google Maps, have introduced local search services that appear to retrieval geographically relevant information using location information of web pages; The research of location-aware text retrieval, which combines both location proximity and text contents in text retrieval, receives much attention [20, 11]. Both commercial search engines and location-aware text retrieval focus on the aspect of “content location”, while local interaction patterns have many benefits to the location-based web services based on the “serving locations” of web pages. With the awareness of the prevalence of common interests among people who are geographically closed together and frequently visit one common web resource or a set of common web resources, businesses are keen to increase their competitive edge by offering geographically tailored contents that reflect the common interests of the geographical region of the web visitors.

As we know, existing web servers typically organize the web pages they host in some hierarchical structures. For example, a commercial web server may organize the pages into different categories such as: “Sports”, “Entertainment”, “Shopping”, “News”, and each category is further decomposed into many subcategories. This static organization is often a source of frustration for the web visitor as they need to perform multiple clicks before they are able to locate the items of their interests. In order to provide better services and increase customer satisfaction, many web servers are looking at customer-centric organization whereby the content of the web pages are customized based on the locations of visitors. It is able to do so because many web servers have already accumulated gigantic log files recording the details of each access such as: the source IP address, the time and duration of access, and the pages visited. Analyzing these log files for geographical-specific common interests among the web visitors is a promising approach to dynamically customized the web structure based on the interests that have been shown by the web visitors in the same geographical region.

In order to achieve this objective, an efficient algorithm that can automatically discover the geographical-specific common interests among the web visitors are needed. In addition, since these interests may change from time to time, it is useful to know what and how these interests change over time. For example, the Asian visitors may tend to click the pages relating to “Tennis” and “Badminton” while the visitors in North America tend to click on the pages relating to “Football” and “Basketball”. In this case, we may conclude that the Asian visitors share the common interests of “Tennis” and “Badminton”, while the North American visitors prefer “Football” and “Basketball”. Furthermore, the interests of the Asian visitor changes from “Tennis” in the months of June- October to “Badminton” in the months of March-July, while that of the North American visitors interests changes from “Football” in the months November-February to “Basketball” in the months April-June. The web serve may vary the services from time to time. This example motivates the development of moving interaction patterns, i.e., geographic-specific interaction pattern chains.

In this chapter, we aim to find geographical-specific interaction patterns in some local regions, and discover changes in the supporting regions (e.g., movement, enlargement and shrinkage) over multiple time points. We design an algorithm called FlexiPROBER that utilizes a quadtree structure to iteratively refine the regions so as to discover the local geographical-specific interaction patterns. We define three important pattern trends, i.e., enlargement, shrinkage and movement of supporting regions, to capture the changes in these patterns and develop an algorithm called MineGIC to discover these changes. Experiment results on both synthetic and real world datasets demonstrate that the proposed approach is effective in mining the local geographical-specific interaction patterns and discover their changes over time.

The rest of the chapter is organized as follows. Section 5.1 gives the preliminaries and problem statement. Section 5.2 introduces the multi-scale influence model. Section 5.3 presents the algorithm FlexiPROBER to mine the geographical-specific interaction pattern on static time frame, and Section 5.4 presents the algorithm MineGIC to discover the pattern changes. Section 5.5 presents the results of experiments to evaluate the proposed algorithm. Finally, we conclude in Section 5.6.

5.1 Preliminaries and Problem Statement

Suppose $\mathcal{P}$ is a 2D spatial plane with dimensions $[0, x_{max}]$ x $[0, y_{max}]$ and $\mathcal{F} = \{f_1, f_2, \ldots, f_n\}$ is a *binary* feature set. Each feature f_i could denote an interest of a web visitor. An object o on plane $\mathcal{P}$ is a tuple $\langle x, y; \mathcal{F}_o\rangle$, where (x, y) denotes the geographical location of the object, and $\mathcal{F}_o$ is a binary feature vector of the object.

Chapter 4 proposes an *influence model* to describe spatial data distribution and measure the interaction among two or more distributions. Based on the model, the influence of an object $o = (x_o, y_o)$ to a point $p = (x_p, y_p)$ is measured by

$$Inf(o,p) = \frac{1}{2\pi\sigma^2} exp\{-\frac{(x_o - x_p)^2 + (y_o - y_p)^2}{2\sigma^2}\}, \tag{5.1}$$

where the σ is the influence deviation, specified by the applications. Combining the influences from all the objects exhibiting the same feature, say f, we obtain an *influence map* of this feature, denoted by $M(f)$. The influence of feature f in point p is denoted by $M(f, p)$. Let $\{o_1, o_2, \cdots, o_m\}$ be a set of objects that has the feature f. The influence of f on a point p is $M(f, p) = \sum_{i=1}^{m} Inf(o_i, p)$. In term of statistics, influence map is a density distribution about influence.

Given the influence maps of a set of features $\{f_1, f_2, \cdots, f_n\}$, and we denote

$$MIN(f_1, f_2, \cdots, f_n; p) = min(M(f_1, p), M(f_2, p), \cdots, M(f_n, p)), \tag{5.2}$$

and

$$MAX(f_1, f_2, \cdots, f_n; p) = max(M(f_1, p), M(f_2, p), \cdots, M(f_n, p)), \tag{5.3}$$

we can determine the degree of *interaction* among these features as follows:

$$I(f_1, f_2, \cdots, f_n) = \frac{\int_{p \in \mathcal{P}} MIN(f_1, f_2, \cdots, f_n; p)dp}{\int_{p \in \mathcal{P}} MAX(f_1, f_2, \cdots, f_n; p)dp}. \tag{5.4}$$

Interaction can measure the similarity amongst influence maps. High interaction value means high similarity, and vice versa. Specially, $I(f_1, f_2, \cdots, f_n) = 1$ if and only if these n features assign exactly the same influence to every point on the plane. Note that using KL-divergence is not suitable here because it cannot be easily generalized to measure the difference among three or more distributions.

The degree of interaction among a set of features can be constrained to a region instead of the whole plane. For example, the interaction of a set of features $\{f_1, f_2, \cdots, f_n\}$

on a region $\mathcal{R} \in \mathcal{P}$, denoted by $I(f_1, f_2, \cdots, f_n; \mathcal{R})$, is

$$I(f_1, f_2, \cdots, f_n; R) = \frac{\int_{p\in\mathcal{R}} MIN(f_1, f_2, \cdots, f_n; p)dp}{\int_{p\in\mathcal{R}} MAX(f_1, f_2, \cdots, f_n; p)dp}. \tag{5.5}$$

With this, the *geographical-specific interaction pattern* (or GIP for short) can be formally defined as follows.

Definition 14. *Given a spatial database containing a set of objects and a feature set $\mathcal{F}$, a region $\mathcal{R} \in \mathcal{P}$, an interaction threshold min_I, a* **geographical-specific interaction pattern** *$GIP = \{f_1, f_2, \ldots, f_n; \mathcal{R}\}$ on $\mathcal{R}$, is the set of features $\{f_1, f_2, \ldots, f_n\} \subseteq \mathcal{F}$ such that $I(f_1, f_2, \ldots, f_n;\mathcal{R}) \geq min\_I$. $\mathcal{R}$ is called the* **support region** *for GIP.*

An interaction pattern GIP is called a k-pattern if it consists of k distinct features $\in \mathcal{F}$, that is, $|GIP| = k$.

With this definition, we can track how the support regions of GIP pattern changes over time. In particular, we are interested in discovering three kinds of changes in this chapter.

- Enlargement. The support regions of a particular interaction pattern expand over two or more continuous time frames;
- Shrinkage. The support regions of a particular interaction pattern shrink over at least two continuous time frames;
- Movement. The support regions of one particular interaction pattern move from one region to a neighboring region over two continuous time slots.

Definition 15. *A geographical interest* **chain***, denoted as $GIC = \langle GIP : R_{t_i} \rightarrow R_{t_{i+1}} \rightarrow \cdots \rightarrow R_{t_j}\rangle$, where GIP is a local geographical-specific interaction pattern and R_{t_i} is the set of support regions of GIP at time t_i, must satisfy one of the following three conditions:*

1. $R_{t_k} \subseteq R_{t_{k+1}}$*, for all k,* $i \leq k \leq j-1$*, is an enlargement chain*

2. $R_{t_k} \supseteq R_{t_{k+1}}$*, for all k,* $i \leq k \leq j-1$*, is a shrinkage chain*

3. $\frac{R_{t_k} \cap R_{t_{k+1}}}{min(R_{t_k}, R_{t_{k+1}})} \geq 0$*, for all k,* $i \leq k \leq j-1$*, is a movement chain*

Problem Statement: Given a spatiotemporal database of point objects over the plane $\mathcal{P}$ and an interest threshold *min_I*, we aim to find the complete set of local geographical-specific interaction patterns on the plane $\mathcal{P}$, and generate all the spatial interaction pattern chains.

5.2 Multi-scale Influence Map

To discover the local interaction patterns, we must first construct the influence map for each feature on the plane $\mathcal{P}$. This influence map must allow for different granularity over different regions in $\mathcal{P}$ in order to highlight interests that are local to small regions.

A quadtree structure [61] is used to facilitate the construction of multi-scale influence maps. Initially, the plane $\mathcal{P}$ is one large cell as the root node of quadtree. All objects on the plane are associated to this root node. We estimate the maximum, minimum and the average influence of all the objects on this cell (the details is elaborated in the next paragraphs). The node split criteria is the gap of maximum and minimum influence. High influence gap implies the skew influence distribution on this cell. In other words, this cell can not capture the precise influence value any more. Thus, a cell will be split into 4 equal sub-cells if its influence gap exceeds an assigned error bound, and the associated objects in this cell are pull down to one or more sub-cells. The process is repeated until no cell partition is required. For example, assuming the objects of feature f_1 distribute coarsely on regions $R13$, $R21$, $R22$, $R23$, $R31$, and the objects of feature f_2 are on regions $R11$, $R12$, $R21$, $R22$, $R32$, $R33$. Figure 5.1 shows the results of implementing the strategy on feature f_1 and feature f_2 respectively.

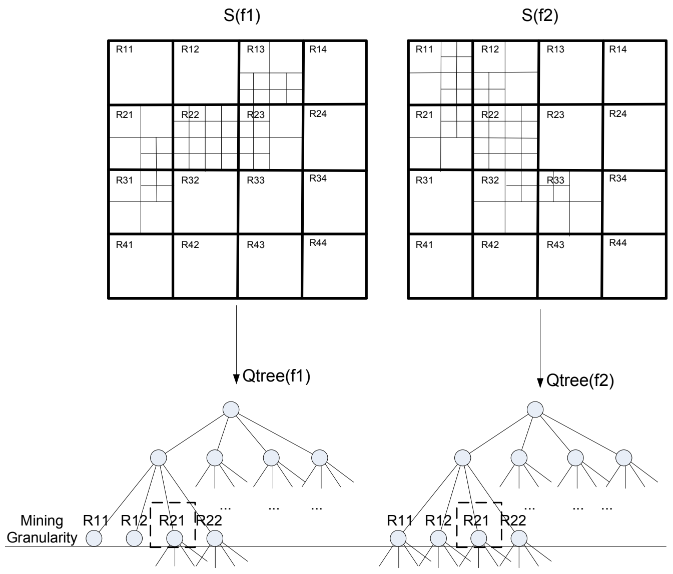


Figure 5.1: Influence Maps and Quadtrees

Please note that the $\mathcal{R}$ in Definition 14 may not be the leaf node in the quadtree. The size and location of R are determined by application interests. For example, we partition the plane into 4×4 buckets in Figure 5.1, and $\mathcal{R}$ can be any one of these buckets.

The computation of the maximum, minimum, and average influences of an object o to a cell G is as follows.

Case 1: Object o is in cell G.

In this case, the minimal distance between o on G is 0, the average distance is determined by the Euclidean distance between o and the center point of G, and the maximal distance is the Euclidean distance between o and the furthest corner of G.

Case 2: Object o is outside cell G.

In this case, the minimum/maximum distance is the Euclidean distance from o to the nearest/furthest corner of the cell$G$, and the average distance is the Euclidean distance between o and the center of G.

Knowing the minimum, average, and maximum distances, we can compute its maximum, average, and minimum influence according to the influence function in Equation 5.1.

With this, the average influence of a feature f on a cell G, denoted by $avg\_inf(f, G)$, is the *summation* of all the average influences from the objects in G with feature f. The minimum (maximum) influence, denoted by $min\_inf(f, G)$ ($max\_inf(f, G)$) are defined similarly.

To determine whether one cell G should be partitioned into 4 sub-cells G^1, G^2, G^3 and G^4, we introduce the notion of scale error ($ScaleErr$).

$$ScaleErr(f, G) = \frac{1}{4} \times \sum_{k=1}^{4} \frac{abs(avg\_inf(f, G) - avg\_inf(f, G^k))}{max(avg\_inf(f, G), avg\_inf(f, G^k))}. \tag{5.6}$$

In our work, $ScaleErr$ is used to measure the average influence change of a cell after splitting. $ScaleErr$ is normalized between 0 and 1. Specially, $ScaleErr = 0$ if no influence change after splitting; otherwise, $ScaleErr > 0$. In general, the more change one cell has, the bigger $ScaleErr$ it obtains.

The computation of $ScaleErr$ is expensive. Suppose there are n objects exerting some influences to a cell, we require a total of $4n$ distance computations. For a quadtree of height h, the worst time complexity is $O(4^h \times n)$. To cut down the cost of constructing the quadtree, we derive an early terminating condition to stop partitioning based on the following two observations.

Observation 1. For any cell G, we have $min\_inf(f, G) \leq avg\_inf(f, G) \leq max\_inf(f, G)$. This property follows from the fact that for any object o, its minimum distance to a cell G is less than its average distance, which in turn, is less than its maximum distance to the cell. By Equation 1, we know the minimum influence of object o on G is less than the average influence which is less than the maximum influence. Summing over all the objects that are associated to G, we have $min\_inf(f, G) \leq avg\_inf(f, G) \leq max\_inf(f, G)$.

Observation 2. Given a feature f, a cell G and its four sub-cells G^1, G^2, G^3 and G^4, $min\_inf(f, G) \leq min\_inf(f, G^k)$ and $max\_inf(f, G) \geq max\_inf(f, G^k)$ where $1 \leq k \leq 4$. This can be proved as follows. For any object instance o_i of feature f, we have $min\_dis(o_i, G) \leq min\_dist(o_i, G^k)$. By Equation 5.1, we conclude $max\_inf(o_i, G)$ $\geq max\_inf(o_i, G^k)$. Summing all object instances of f and applying the transitivity property of inequality, we have $max\_inf(f, G) \geq max\_inf(f, G^k)$. The proof for $min\_inf(f, G)$ $\leq min\_inf(f, G^k)$ is similar.

Theorem 3. *Given a feature f, a cell G, and error bound E, we say that G does not require further partitioning if both of the following conditions hold*

1. $\frac{min\_inf(f,G)}{avg\_inf(f,G)} > 1 - E$;
2. $\frac{max\_inf(f,G)}{avg\_inf(f,G)} < 1 + E$.

PROOF: Suppose G^k is one of the sub-cells of G. From Observation 1 and 2, we know $min\_inf(f, G) \leq min\_inf(f, G^k) \leq avg\_inf(f, G^k) \leq max\_inf(f, G^k) \leq max\_inf(f, G)$.

Case I: $avg\_inf(f, G) \leq avg\_inf(f, G^k)$.

By definition,

$ScaleErr(f, G) = \frac{1}{4} \times \sum_{k=1}^{4} \frac{avg\_inf(f,G^k) - avg\_inf(f,G)}{avg\_inf(f,G^k)}$

$\leq \frac{1}{4} \times \sum_{k=1}^{4} \frac{max\_inf(f,G) - avg\_inf(f,G)}{avg\_inf(f,G)}$ Condition (2)

$\leq \frac{1}{4} \times \sum_{k=1}^{4} (1 + E - 1) = E$;

Case II: $avg\_inf(f, G) \geq avg\_inf(f, G^k)$.

Again, we have

$ScaleErr(f, G) = \frac{1}{4} \times \sum_{k=1}^{4} \frac{avg\_inf(f,G) - avg\_inf(f,G^k)}{avg\_inf(f,G)}$

$\leq \frac{1}{4} \times \sum_{k=1}^{4} \frac{avg\_inf(f,G) - min\_inf(f,G)}{avg\_inf(f,G)}$ Condition (1)

$\leq \frac{1}{4} \times \sum_{k=1}^{4} (1 - (1 - E)) = E$.

□

Theorem 3 provides the early termination condition for unnecessary cell partitioning. With this, we give the details of the multi-scale influence map construction in Algorithm 8 $BuildQTree(f, \sigma)$. The algorithm start with a root node (Line 1) which is initially associated with all objects (Lines 2-4). A function $Split(\cdot)$ is called in Line 5 to split the root node and its sub-node recursively.

Algorithm 8: $BuildQTree(f, \sigma)$

```
input : $\{o_1, \ldots, o_n\}$: $n$ objects of feature $f$;
        $\sigma$: the standard deviation of influence function.
output: Quadtree $T$.
1  initialize a root node $G$ of Quadtree $T$;
2  for each object $o_i$, $1 \leq i \leq n$ do
3      append $o_i$ into $G.obj\_list$;
4  call $Split(G, f, \sigma)$;
5  return $T$;
```

Algorithm 9: $Split(G, f, \sigma)$

```
1  if $\frac{min\_inf(f,G)}{avg\_inf(f,G)} > 1 - E$ AND $\frac{max\_inf(f,G)}{avg\_inf(f,G)} < 1 + E$ then
2      Exit;
3  Split $G$ into four sub-nodes $G^1$, $G^2$, $G^3$ and $G^4$;
4  for each object $o_i \in G.obj\_list$ do
5      for each sub-node $G^k$ do
6          build a region of $G^k$ of radius $\sigma$;
7          if $o_i$ falls into region of $G^k$ then
8              append $o_i$ into $G^k.obj\_list$;
9  if $ScaleErr(f, G) > E$ then
10     for each sub-node $G^k$ do
11         $Split(f, G^k)$;
```

Algorithm 9 gives the details of function $Split(\cdot)$. Line 1 is an early termination condition. Line 4 splits the area of node into four parts. Lines 5-12 distribute the objects into four sub-nodes. If the influence error of the node and its sub-nodes less than 0.05, we terminate further split operation of sub-nodes (Line 13); otherwise, we continue to split each sub-node.

5.3 FlexiPROBER

Having built the multi-scale influence maps for all the features, the task now is to find the geographical-specific interaction patterns, GIP in short, efficiently.

Recall, a GIP in region $\mathcal{R}$ is the set of features whose interaction value among the features in $\mathcal{R}$ is not less than the threshold min_I. Consider a pair of features f_1 and f_2 with influence maps as shown in Figure 5.2(i). Focusing on the region R_{21}, we note that both maps have 7 cells but are of different granularity. Computing the interaction value of f_1 and f_2 on $\mathcal{R}$ requires both maps to be of the same granularity. This is achieved by refining the maps till all corresponding cells have the same granularity. Figure 5.2(ii), shows the results after the refining process. Note that this refining process is relatively inexpensive as there is no need to perform object allocation from the parent node to the child nodes. In the final step, for each common subregions of $\mathcal{R}$, we compute the interaction among the features as follows.

$$I(f_i, f_j; R) = \frac{\sum_{G \in R}(min(avg\_inf(f_i, G), avg\_inf(f_j, G)) \times area(G))}{\sum_{G \in R}(max(avg\_inf(f_i, G), avg\_inf(f_j, G)) \times area(G))} \tag{5.7}$$

Continuing with our example, Figure 5.2(iii) shows that the final interaction of f_1 and f_2 on region R_{21} is $\frac{5\times4+(50+50+50+12)+10\times4+(30+30+30+30)}{25\times4+(50+60+65+50)+15\times4+(120+120+150+136)} = \frac{342}{911} = 0.38$.

To generate the set of GIPs, we adopt the level-by-level candidate pattern generation procedure. A k-pattern candidate is generated from two (k-1)-patterns if they share a common k-2 prefix [65]. For GIPs, we prove that the GIPs satisfy the Min-Max theorem as stated below.

Theorem 4. *(Min-Max Theorem) Given two influence maps on $\mathcal{R}$, and the interaction threshold min_I, $\{f_i, f_j\}$ cannot be a GIP on $\mathcal{R}$ if one of the following conditions hold,*

1. $\frac{max\_inf(f_i,R)}{min\_inf(f_j,R)} < min\_I$, *or*

2. $\frac{max\_inf(f_j,R)}{min\_inf(f_i,R)} < min\_I$,

where $max\_inf(f_i, \mathcal{R})$ ($min\_inf(f_i, \mathcal{R})$) is the maximal (minimal) influence of f_i on $\mathcal{R}$.

PROOF: We show the proof for Condition (1). The proof for Condition (2) is similar.

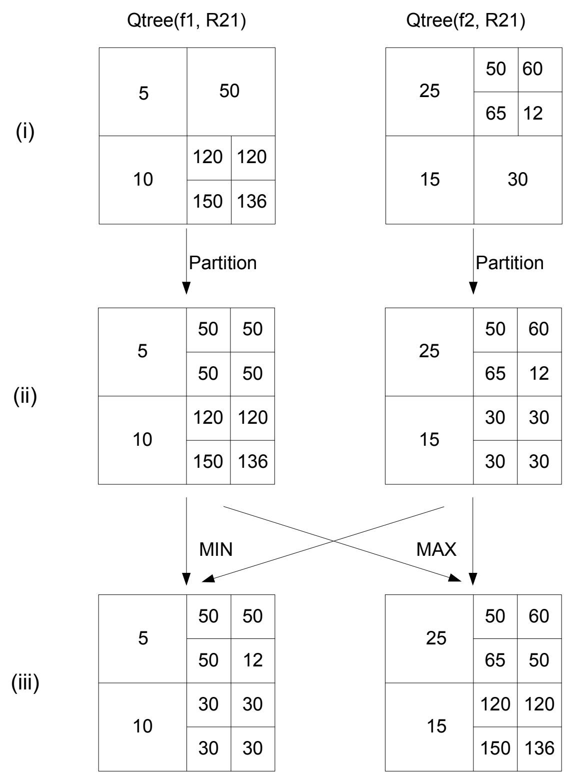


Figure 5.2: Interaction of f_1 and f_2 on Region R_{21}

Suppose Condition (1) holds, since $min\_I \leq 1$, we can infer that $min(avg\_inf(f_i, G), avg\_inf(f_j, G)) = avg\_inf(f_i, G)$, and $max(avg\_inf(f_i, G), avg\_inf(f_j, G)) = avg\_inf(f_j, G)$, where G is a subregion in $\mathcal{R}$.

By Equation 5.7, $I(f_i, f_j; R) = \frac{\sum_{G \in R}(avg\_inf(f_i,G) \times area(G))}{\sum_{G \in R}(avg\_inf(f_j,G) \times area(G))} \leq \frac{\sum_{G \in R}(max\_inf(f_i,R) \times area(G))}{\sum_{G \in R}(min\_inf(f_j,R) \times area(G))}$ $= \frac{max\_inf(f_i,R) \times \sum_{G \in R} area(G)}{min\_inf(f_j,R) \times \sum_{G \in R} area(G)} \leq min\_I$.

□

This theorem enables a large number of candidates to be pruned, resulting in a highly efficient algorithm called *FlexiPROBER*. Details of *FlexiPROBER* are given in Algorithm 10. Lines 1-3 construct the quadtree for each feature by calling $BuildQTree(\cdot)$. For each region R, Lines 5-14 mine the complete set of GIPs with the call to procedure $Apriori\_gen(\cdot)$ in Line 8, and compute the interaction values in Line 10.

Procedure $Apriori\_gen(\cdot)$ (shown in Algorithm 11) generates the k-pattern candidates from the (k-1)-pattern sets. Line 5 combines two (k-1)-patterns if they share

Algorithm 10: FlexiPROBER

```
input : $D$: the spatial database;
        $RF$: all features in $D$;
        $min\_I$: interaction threshold;
        $\sigma$: the standard deviation of influence function;
        $h$: the mining granularity;
output: $P$: the set of interaction patterns.
1  for each feature $f_i \in RF$ do
2      call $BuildQTree(f_i, \sigma)$;
3  $P = \emptyset$;
4  for each cell in the h-th level of quadtree, $\mathcal{R}$ do
5      $GIP_1 = QTree(f_i, \mathcal{R})$, where $f_i \in RF$;
6      for $k = 2; P_{k-1} \neq \emptyset; k++$ do
7          $C_k = Apriori\_gen(GIP_{k-1}, min\_I, \mathcal{R})$;
8          //Apriori property
9          for each candidate $c \in C_k$ do
10             compute the interaction of $c$;
11         $GIP_k = \{c \in C_k | c.interaction \geq min\_I\}$;
```

common k-2 prefix. Lines 6-7 prune the candidate pattern using the MIN-MAX theorem. Lines 9-11 compute the minimal influence and maximal influence of the candidate pattern and add it to the candidate pattern sets.

5.4 Discovering Interaction Patterns Changes

Suppose D_1, D_2, …, D_q correspond to the datasets at time t_1, t_2, …, t_q. We can mine GIPs from each of these datasets independently as described in Section 4. Having generated the GIPs for each time point, we next consider how to detect interesting changes in these patterns over time. Note that Enlargement and Shrinkage chains can be extended from Movement chains, because they are the special cases of Movement chains if $\frac{R_{t_k} \cap R_{t_{k+1}}}{R_{t_k}} = 1$ or $\frac{R_{t_k} \cap R_{t_{k+1}}}{R_{t_{k+1}}} = 1$. Hence, we only present the approach to discover the Movement chains.

Given a pattern GIP, we use a bitmap structure to indicate its support regions at

Algorithm 11: $Apriori\_gen(GIP_{k-1}, min\_I, \mathcal{R})$

```
1  for each pattern $GIP_{p_1} \in GIP_{k-1}$ do
2      for each pattern $GIP_{p_2} \in GIP_{k-1}$ do
3          if $GIP_{p_1}$ and $GIP_{p_2}$ have the identical prefix $k-2$ features then
4              //$f$ is a new feature;
5              $f = GIP_{p_2}[k-1]$;
6              $c = GIP_{p_1} \cup GIP_{p_2}$;
7              if $\frac{max\_inf(p_1,R)}{min\_inf(f,R)} < min\_I$ OR $\frac{max\_inf(f,R)}{min\_inf(p_1,R)} < min\_I$ then
8                  //MIN-MAX Theorem;
9                  delete $c$;
10             else
11                 $c.min\_inf = MIN(GIP_{p_1}.min\_inf, f.min\_inf)$;
12                 $c.max\_inf = MAX(GIP_{p_2}.max\_inf, f.max\_inf)$;
13                 add $c$ to $C_k$;
14 return $C_k$;
```

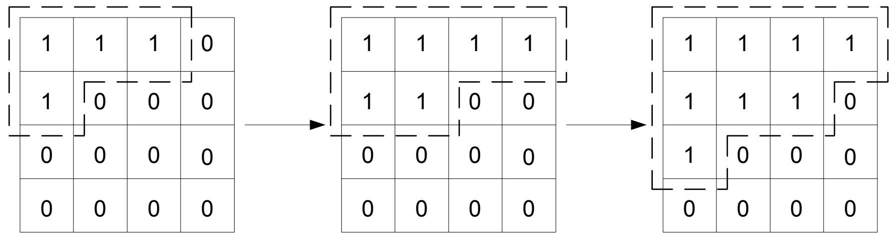
time point t_i, i.e. R_{t_i} in Definition 15. A bit is set to 1 if the corresponding region supports the pattern. Figure 5.3 shows examples of 4x4 bitmap structures where the GIC_1 demonstrates an enlargement chain, starting from 4 regions at time t_0 to 6 regions at time t_1 to 8 regions at time t_2; while GIC_2 is an example of a movement chain where the 4 support regions are shifted in t_1 to t_3.

A naive method to discover enlargement, shrinkage and movement chains is to use FlexiPROBER to generate the GIPs for each time frame. For each GIP, we check the condition for all consecutive time frames to determine whether the GIP is an enlargement, shrinkage or movement chain. This approach involves many unnecessary tests.

We observe that a GIP can participate in an enlargement, shrinkage or movement chain only if its sub-patterns occurs in some common time intervals with overlapping regions. We introduce the notion of a *spatiotemporal join* to capture this concept of common time intervals with overlapping regions.

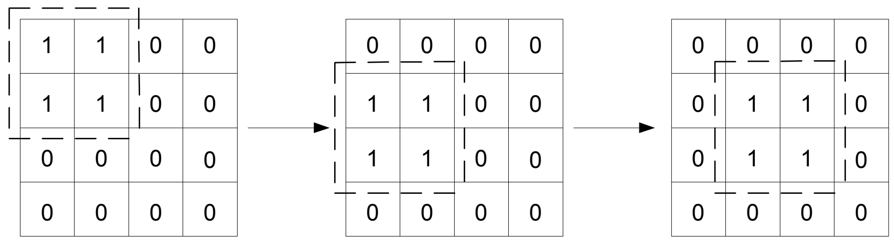
Let $PMap(GIP, t)$ denote the bitmap that indicates the support regions of GIP at time t. We define the *spatiotemporal join* of two chains GIC_1 and GIC_2, denoted as

$$GIC_1 = \langle GIP_1 : [t_0, 1110100000000000] \rightarrow [t_1, 1111110000000000] \rightarrow [t_2, 1111111010000000] \rangle$$



(a) Enlargement chain

$$GIC_2 = \langle GIP_2 : [t_1, 1100110000000000] \rightarrow [t_2, 0000110011000000] \rightarrow [t_3, 0000011001100000] \rangle$$



(b) Movement chain

Figure 5.3: Examples of pattern chains

$$\langle GIP_1 \bowtie GIP_2 : [t_1, 1100110000000000] \rightarrow [t_2, 0000110010000000] \rangle$$

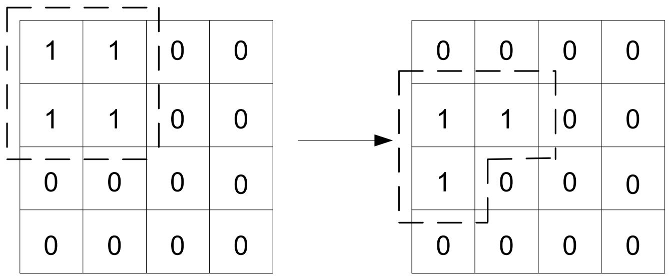


Figure 5.4: Spatiotemporal join GIC_1 and GIC_2

$GIC_1 \bowtie_{st} GIC_2$, as follows.

For two interest chains,

$$GIC_1 = \langle GIP_1 : PMap(GIP_1, t_i) \rightarrow \cdots \rightarrow PMap(GIP_1, t_j)\rangle$$

$$GIC_2 = \langle GIP_2 : PMap(GIP_2, t_m) \rightarrow \cdots \rightarrow PMap(GIP_2, t_n)\rangle$$

let $[t_k, t_l] = [t_i, t_j] \cap [t_m, t_n]$, the spatiotemporal join is defined as

$$GIC_1 \bowtie_{st} GIC_2 = \langle GIP_{\bowtie} : PMap(GIP_{\bowtie}, t_k) \rightarrow \cdots \rightarrow PMap(GIP_{\bowtie}, t_l)\rangle$$

where $GIP_{\bowtie} = GIP_1 \bowtie GIP_2$ (see Algorithm 11), and for each $t \in [t_k, t_l]$, $PMap(GIP_{\bowtie}, t) = PMap(GIP_1, t) \cap PMap(GIP_2, t)$. For example, the spatiotemporal join of GIC_1 and GIC_2 in Figure 5.3 is shown in Figure 5.4.

The Apriori-like property exists in the chain, stating that: If a chain of P from t_i to t_j, $P : R_{t_i} \rightarrow R_{t_{i+1}} \rightarrow \cdots \rightarrow R_{t_j}$, is a Movement chain, then any subpattern of P, P', also has a Movement chain $P' : R'_{t_i} \rightarrow R'_{t_{i+1}} \rightarrow \cdots \rightarrow R'_{t_j}$, where $R'_{t_i} \supseteq R_{t_i}, \ldots, R'_{t_j} \supseteq R_{t_j}$. It is the foundation of our next Algorithm MineGIC to discover the interesting movement chains by level-wise mining.

The pseudocode of MineGIC is given in Algorithm 12. Lines 1-3 read the sub-datasets and build the independent quadtrees as discussed in Section 4. Lines 4-6 initializes each feature as a 1-pattern chain. Lines 7-27 describe the level-wise candidate generation process. Line 8 picks two chains from the (k-1)-pattern set. Line 9 performs the spatiotemporal join of the two chains if they have (k-2) common features, and obtains a candidate chain C of one k-pattern. Line 10 initializes a queue for this candidate chain C and Line 11 push the first $PMap$ of chain C into this queue. Lines 12-22 detail an iterative $PMap$ comparison process for each time frame with its previous one. Each comparison consists of two phases. The first phase is the determination of the overlapping regions, shown in Lines 13-16. $FlexiPROBER$ is called to compute the interaction

Algorithm 12: MineGIC

```
input  : The dataset from $t_0$ to $t_e$, the necessary parameters as Algorithm 10
output: The Movement chains $GIC$
1  for each feature $f_i \in RF$ and each time $t_j$, $0 \leq j \leq e$ do
2      call $BuildQTree(f_i, \sigma)$;
3  $GIC = \emptyset$;
4  $GIC_1 = \{f_i : R_{t_0} \rightarrow \ldots \rightarrow R_{t_e}\} | f_i \in RF\}$;
5  $GIC = GIC + GIC_1$;
6  for $k = 2; GIC_{k-1} \neq \emptyset; k++$ do
7      for two chains $GIC_1, GIC_2 \in GIC_{k-1}$, where $GIC_1$ and $GIC_2$ have the
       common (k-2) prefix features do
8          $GIC = GIC_1 \bowtie_{st} GIC_2$;
9          Initialize a queue $Q$;
10         Push $PMap(GIC.t_{begintime})$ into $Q$;
11         for $t_j = C.begintime + 1$ to $GIC.endtime$ do
12             for $\mathcal{R}$, $\mathcal{R}$=1 in element of $PMap(GIC.t_j) = 1$ do
13                 Compute the interaction of $GIC$ in the location $\mathcal{R}$;
14                 Let $\mathcal{R}$=0 in $PMap(GIC.t_j)$, if $I(GIC; \mathcal{R}) < min\_I$;
15             Push $PMap(GIC.t_j)$ into $Q$;
16             if $PMap(GIC.t_j) \cap PMap(GIC.t_{j-1}) = \emptyset$ then
17                 Pop the first $PMap$ in $Q$ to $PMap(GIC.t_{j-1})$ as a chain $GIC$;
18                 $GIC_K = GIC_K + GIC$;
19     $k++$;
```

of this k-pattern in the overlapped regions. If the interaction is less than min_I threshold, this region does not support the k-pattern, and the corresponding $PMap$ bit is set to 0. In the second phase, shown in Lines 18-21, we compare the $PMap$ of the current time frame with the previous one in queue and remove those that do not have any overlapped regions with the previous time frame.

5.5 Experimental Studies

In this section, we examine the performance of FlexiPROBER on both synthetic and real world datasets. We also compare the performance of MineGIC with the naive approach. All algorithms are implemented in C++ and the experiments are carried out on a Pentium IV PC with 3GHz CPU and 1GB memory, running Windows XP.

Synthetic Datasets: We use the synthetic data generator in [77] to generate the spatial datasets. It generates objects of different features that are close to each other in some regions. The target plane is $[0, 8192] \times [0, 8192]$ and the features have different density distributions on this plane. The datasets are named using the convention Data-(f)-(N) to indicate the number of features f and total objects N. For example, DATA-8-50k is a dataset which contains 8 features and a total of 50,000 objects.

Real World Datasets: We test our algorithm on a log file dataset of the web server of an academic institute. We capture four weeks of web log files, from October 23 to November 23 in the year 2006, which record the accesses to the web site of the School of Computing, National University of Singapore (http:// www.comp.nus.edu.sg). This web log consists of the IP addresses of visitors to the website. On average, the number of accesses are about 20,000 per day, after excluding repeated IPs and dirty data. We use the IP locator software GeoLyzer [1] to identify the visitor's geographical location in terms of the longitude and latitude coordinates in the world map. In addition, the log file also captures the pages accessed by the visitor over a period of four weeks. We categorize the visited web pages into 15 features as shown in Table 5.1, where "PP" is the abbreviation for "Personal Pages". For example, all web pages of "Graduate Program" are labelled as feature f_1, and so on.

5.5.1 Effectiveness

We show the interesting chains discovered by MineGIC on the web log dataset. We first partition the world map into 8×8 cells as shown in Figure 5.6. We use [X-id, Y-id] to refer to a cell in the plane. For example, cell [1,5] contains the west coast and mid-west

[1]http://www.geobytes.com/GeoLyzer.htm

Table 5.1: Features in Web log Real Dataset

FID	Target on	FID	Target on
f_1	Graduate Program	f_9	PP on Media research
f_2	Undergraduate Program	f_{10}	PP on System research
f_3	Research	f_{11}	PP on Software research
f_4	Computer Science Dept.	f_{12}	PP on Electronic Commerce research
f_5	Information System Dept.	f_{13}	PP on Information Privacy research
f_6	PP on AI research	f_{14}	PP on Knowledge Management research
f_7	PP on Bio research	f_{15}	PP on Virtual Communities research
f_8	PP on DB research		

of U.S while cell [4, 5] covers Europe. In this experiment, *min_I* is 0.4 and σ is 1.

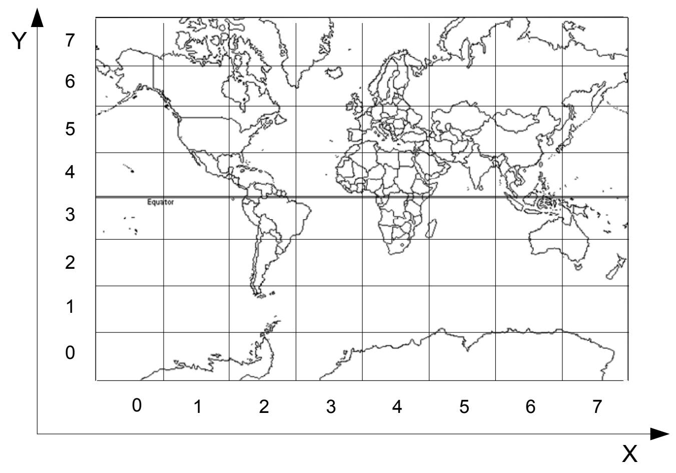
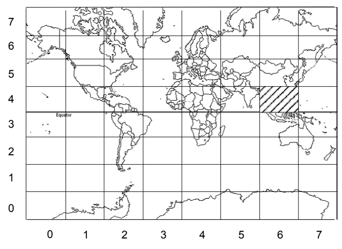
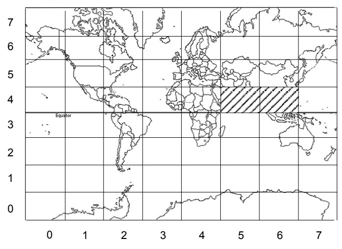
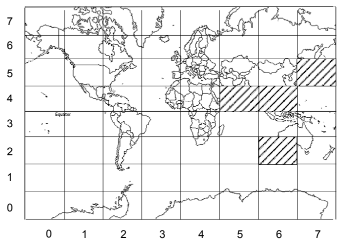


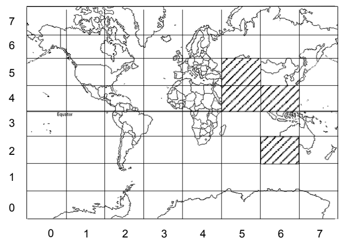
Figure 5.5: The 8×8 bitmap over the world map

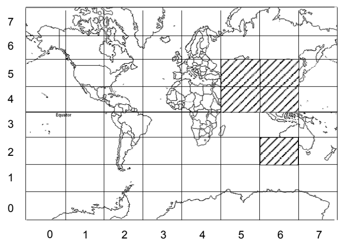
Figure 5.6 shows the trend of $\{f_4, f_8\}$ where shadow areas indicate the support regions. This chain can be interpreted as follows: On Monday Oct 23, web visitors from South China (cell [6,4]) showed an interest in the database research (f_8). The next day, the interest in database research has expanded to India (cell [5,4]), and subsequently to Australia (cell [6,2]) and Japan (cell[7,5]). On Saturday Oct 28, we note a decline in the interest with only visitors from China and India accessing the web pages. The trend of $\{f_4, f_8\}$ is intuitive as it captures the typical access patterns over a week where the interest emerges on Monday and eventually declines as the weekend approaches.


(a) Oct.23 (Monday)


(b) Oct.24 (Tuesday)


(c) Oct.25 (Wednesday)


(d) Oct.26 (Thursday)


(e) Oct.27 (Friday)

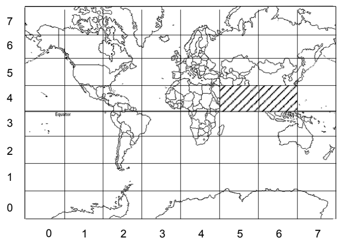
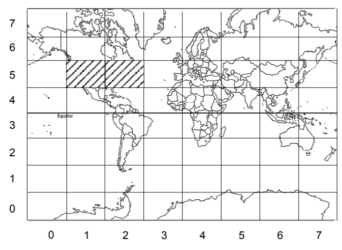
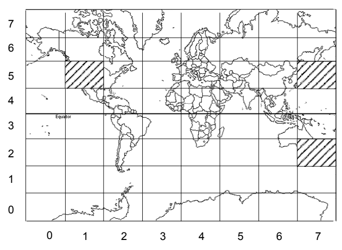

(f) Oct.28 (Saturday)

Figure 5.6: The chain of pattern $\{f_4, f_8\} = \langle\{f_4, f_8\} : ([6,4]) \rightarrow ([6,4][5,4]) \rightarrow ([6,4][5,4][6,2][7,5]) \rightarrow ([5,5][5,4][6,4][6,2]) \rightarrow ([5,5][5,4][6,4][6,2][6,5]) \rightarrow ([5,4][6,4])\rangle$


(a) Oct.27 (Friday)


(b) Oct.28 (Saturday)

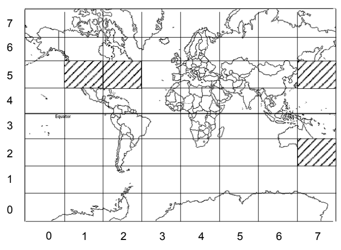

(c) Oct.29 (Sunday)

Figure 5.7: The chain of pattern $\{f_1, f_4, f_5\} = \langle\{f_1, f_4, f_5\} : ([1,5][2,5]) \rightarrow ([1,5][7,5][7,2]) \rightarrow ([1,5][2,5][7,5][7,2])\rangle$

Another interesting chain $\{f_1, f_4, f_5\}$, indicated again by shadow areas, is given in Figure 5.7. This chain demonstrates a growing interest of web visitors from various places in the graduate program offered by the computer science and information systems departments. On Friday, visitors in the United States (cells [1,5] and [2,5]) accessed these web pages. On Saturday, we observe additional visitors from Japan (cells [7, 5]), Australia and New Zealand (cells [7, 2]).

5.5.2 FlexiPROBER versus PROBER

In this set of experiments, we study the efficiency and scalability of FlexiPROBER. We run the experiments on both synthetic and real-world datasets. The baseline algorithm is PROBER [65], which utilizes a uniform grid framework to model the influence. In PROBER, each influence map is a matrix of $n \times n$ where n is a user-defined granularity of the plane $\mathcal{P}$.

Efficiency

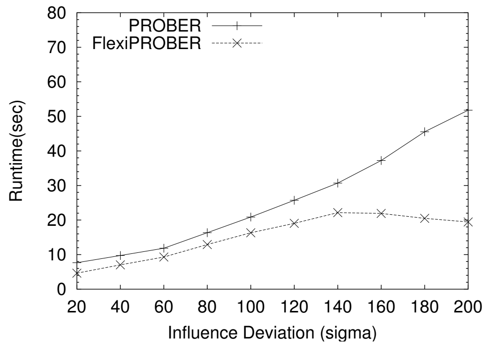
We define the skewness of a dataset as follows:

$$skew_f = \frac{Area\ of\ feature\ f}{Total\ area}.$$

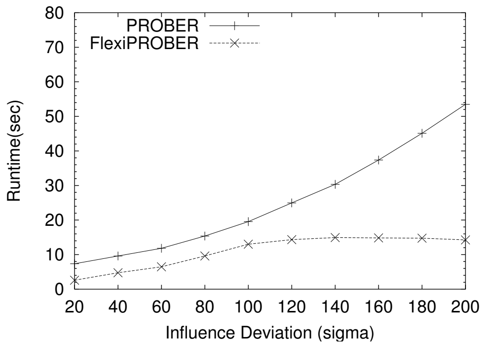
For example, $skew_{f1}$=0.6 indicate that the objects of feature f_1 cover 60% of the plane. For uniform distribution where objects are distributed equally throughout the plane, $skew_{f1}$=1.0.

Figure 5.8 gives the results of PROBER and FlexiPROBER on the synthetic datasets of varying skewness. The three plots in Figure 5.8 show that the runtimes of both PROBER and FlexiPROBER increase as σ increases, but the runtime of PROBER increases faster than that of FlexiPROBER. This is expected as FlexiPROBER employs early termination condition to avoid unnecessary candidate generation.

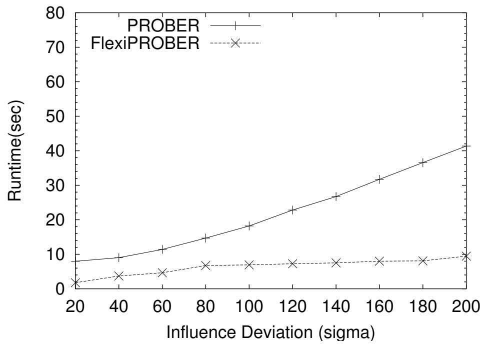
The three plots in Figure 5.8 also show that gap between PROBER and FlexiPROBER is increasing as the skewness increases. This result indicates that the data distribution affects the efficiency of FlexiPROBER. For data of skew distribution, FlexiPROBER imposes fine granularity on the region of high data density and coarse granularity on the region of low data density. In this case, the size of influence map is adaptive to the data distribution, so both space and computational complexities decrease. However, in the uniform distribution, FlexiPROBER is similar to PROBER because FlexiPROBER imposes the same granularity over the plane.



(a) Data-8-50k, uniform



(b) Data-8-50k, skew = 0.6



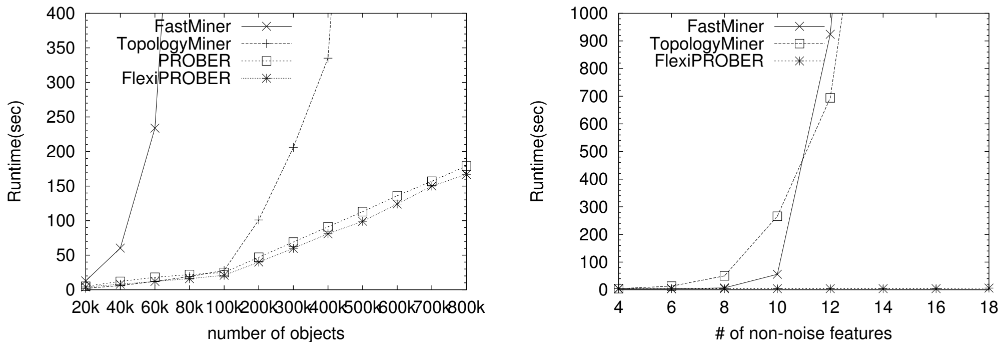
(c) Data-8-50k, skew = 0.1

Figure 5.8: Efficiency of building influence maps

Scalability

We also examine the scalability of FlexiPROBER. We fix the number of features to be 10 and generate twelve datasets Data-10-{20k, 40k, 60k, 80k, 100k, 200k, . . ., 800k} with uniform distribution of objects. The results are shown in Figure 5.9(a). We observe that, as database size increases, the runtimes of PROBER and FlexiPROBER increase linearly, and the runtimes of FastMiner and TopologyMiner increase exponentially. This result shows that PROBER and FlexiPROBER are more scalable than FastMiner and TopologyMiner. This is expected because PROBER and FlexiPROBER do not rely on expensive spatial join. We also observe that FlexiPROBER is slightly faster than PROBER for each setting of database size. This is expected because FlexiPROBER perform better than PROBER in the situation where the data distribution is skewed.

We set the database size at 20k instances and generate eight datasets Data-{4, 6, 8, 10, 12, 14, 16, 18}-0-50-20k by varying the number of collocated features. Figure 5.9(b) shows the results. We observe that both FastMiner and TopologyMiner do not scale well when the number of features increases. On the other hand, the runtime of FlexiPROBER does not increase greatly because the multi-granularity mechanism is able to compute and compare the influence space efficiently.

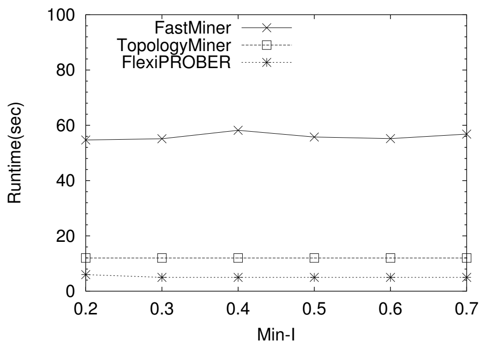


(a) Effect of DB size (b) Effect of non-noise feature number

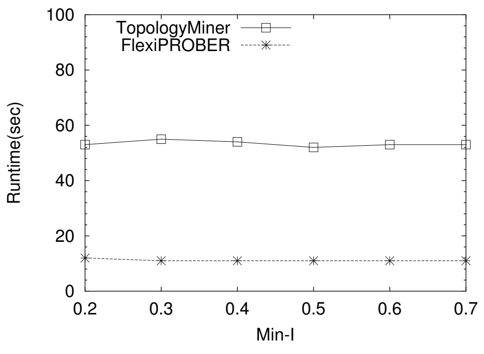
Figure 5.9: Scalability of FlexiPROBER

Sensitivity

Next, we examine the effect of parameter *min_I* on two test datasets Data-6-2-75-50k and Data-8-2-75-200k. Figure 5.10 shows the experimental results of the effect of *min_I* on runtime. We observe that in all cases, the runtime of the three algorithms are hardly affected by the increase in *min_I*. We also note that FlexiPROBER outperforms the other two algorithms on both datasets. In particular, FastMiner's runtime in the second dataset is beyond the maximum scale in 5.10(b). Both plots in Figure 5.10 demonstrate that the influence based algorithms are not sensitive to the parameter *min_I*, because the mining cost is dominated by the cost of building influence maps.



(a) Data-6-2-50k



(b) Data-8-2-100k

Figure 5.10: Effect of *min_I*

5.5.3 MineGIC versus Naive Approach

We compare the efficiency of MineGIC with the naive approach described in Section 5.4 on the NUS web log data. 15 time frame data are selected (from Oct 23 to Nov 6) from the NUS web log dataset. Figure 5.11 shows that MineGIC performs much better than the naive approach. The latter is not scalable as FlexiPROBER must be re-executed for each time frame. In contrast, MineGIC focuses only on those patterns that have the potential to be GICs. This results in an efficient pruning of a large number of candidates.

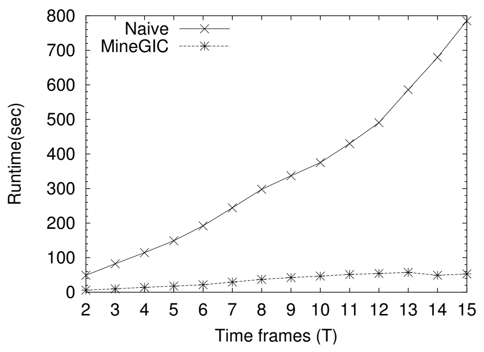


Figure 5.11: Efficiency of MineGIC

5.6 Summary

In this chapter, the grid based influence model is extended to the quadtree based influence model, which is adaptive to the data distribution and flexible to support local interaction pattern mining. We obtain the influence map of each feature and compute the degree of their interactions. Those features with high degrees of interaction are the geographical-specific interaction patterns. Based on the quadtree, we propose a multi-scale FlexiPROBER algorithm to discover the geographical-specific interaction patterns. Further, we design the algorithm MineGIC to efficiently mine all the enlargement, shrinkage, and movement chains of geographical-specific interaction patterns. MineGIC is applied in the web click data to discover the geographical interest changes of the web visitors. Experiment results on synthetic and real world datasets demonstrate that FlexiPROBER and MineGIC are both efficient and scalable and can find meaningful geographical-specific interaction patterns at one time frame and over multiple time frames.

Chapter 6

Mining Duration-Aware Trajectory Patterns in Moving Object Data

Besides biological data and snapshot data, another important type of spatiotemporal data is moving object data. Moving object data is more and more popular due to the rapid spread of GPS system and the development of tracking techniques. This has led to the wide research interests in knowledge discovery in moving object data [25, 42, 35]. In this chapter, we focus on the mining of trajectory patterns in moving object data and the application of these trajectory patterns in trajectory classification.

As a crucial model in trajectory data analysis, *trajectory classification* is an important research problem. Assume each trajectory in the trajectory database has a class label. Trajectory classification is the process of predicting the class labels of moving objects based on their trajectories and other features.

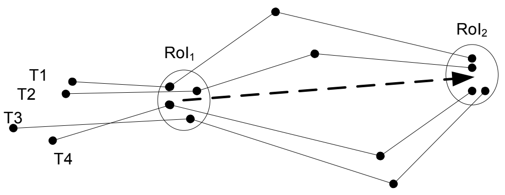
The ability to classify trajectories is useful in many real world applications. In meteorology, a trajectory classifier can predict the intensity and scale of an approaching hurricane, so that precautionary actions can be carried out in advance. In homeland security, it is reported that more than 160,000 vessels are travelling in the United States' waters [45], and an anomaly trajectory detection classifier that can evaluate the vessels' behaviors and highlight suspicious vessels for further monitoring is highly desirable.

Existing work on trajectory classification [42] selects the regions and representative trajectories as the features for classification. Regions are mined based on the spatial distribution of trajectories, and representative trajectories are mined based on the shapes of trajectories, as shown in Figure 6.1(b). However, it does not take the duration of the trajectories into consideration in differentiating the objects that move at different speeds. For example, the speed at which a tropical hurricane passes the Gulf of Mexico is an important criterion in classifying its scale and intensity. Classifiers, that look only at the spatial distributions and movement directions of hurricanes but ignore the moving speeds, are unable to accurately classify the intensities of the hurricanes.

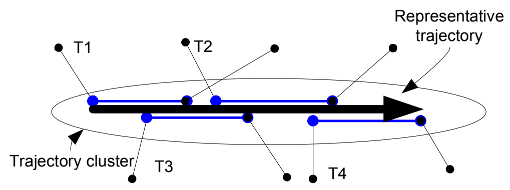
We introduce duration-sensitive region rules to highlight regions where there is a differentiating number of trajectories of one class passing through them taking into account the time spent by these trajectories in the regions. We propose a top-down space partition approach that recursively partitions a region into smaller regions. The partitioning criterion is based on the information gain measure. The result is a set of highly discriminative regions.

We also introduce the notion of speed-differentiating path rules to capture the actual movement paths and movement speeds. A speed-differentiating path rule is simply a sequence of object locations with an associated duration time between consecutive pairs of locations. Discovering speed-differentiating path rules from a trajectory database is challenging. Ideally, a path rule should summarize the movement, direction, and speed of a group of similar moving objects such that the distances between the actual trajectories and speeds of these moving objects to the path rule are minimized. To achieve this, we need to adaptively vary the granularity of regions and duration intervals as we perform our mining. Having a pre-determined granularity for regions and duration intervals is undesirable because if the granularity is too coarse, it will lead to a small number of path patterns which is not enough to build an accurate classifier. On the other hand, if the granularity is too fine, it will lead to a large number of path rules, resulting in overfitting.

Trajectory patterns [25] are not speed-differentiating path rules. First, trajectory patterns are not the actual movement paths. For example, Figure 6.1(a) illustrates a trajectory pattern discovered from four objects T_1, T_2, T_3 and T_4, where the four objects do not follow the same path from RoI_1 to RoI_2. We cannot compute the Euclidean distance of test trajectory and trajectory patterns to measure their similarity. Second, trajectory patterns are very coarse and do not have discriminative power for accurate classification.



(a) Trajectory patterns in [25]



(b) Trajectory clusters in [43]

Figure 6.1: Existing patterns

Besides the need for a scheme that varies the granularity levels of regions and duration intervals adaptively, a second challenge is the high computational complexity in generating path rules from the trajectory dataset. To overcome this, we design an efficient algorithm which can mine discriminative speed-differentiating path patterns and prune undesirable path patterns as soon as possible. First, we summarize the trajectory database in the form of a trajectory network with the appropriate granularity. The level of granularity is controlled and measured by the Minimum Description Length (MDL) gain. Based on the trajectory network, we design a path pattern tree to enumerate the candidate path patterns, and mine the top-k covering path rules.

Two classifiers are built. The first is constructed by transforming the trajectories into score vectors and utilizing an existing classification techniques on these score vectors.

The second is a k-NN based classifier which predicts the class labels of trajectories by the top k highest score rules. Experiment results on three real-world datasets show that classifiers that are built based on both duration-sensitive region rules and speed-differentiating path rules can achieve higher accuracy compared to classifiers that do not take the duration information into account.

The remainder of this chapter is organized as follows. Section 6.1 and 6.2 gives the preliminaries and the problem statement. Section 6.3 gives the overview of our solution. Section 6.4 presents the region partition algorithm to discover region rules. In Section 6.5, we introduce the notation of trajectory network and the training algorithm to obtain a trajectory network, and we introduce the path pattern tree and the top-k covering path rule mining algorithm. In Section 6.7, we evaluate our algorithms real-world datasets. Finally, we conclude our work in Section 6.8.

6.1 Preliminaries

A trajectory T is a time-ordered sequence of sampling points $\langle p_1, t_1\rangle \langle p_2, t_2\rangle \ldots \langle p_N, t_N\rangle$, where p_1, p_2, ..., p_N is a sequence of moving object locations corresponding to sampling time $t_1, t_2, \ldots, t_N$. A trajectory T which is sampled from t_1 to t_N can also be represented as $T[t_1{:}t_N]$. We say that a trajectory $T[t_i{:}t_j] = \langle p_i, t_i\rangle \langle p_{i+1}, t_{i+1}\rangle \ldots \langle p_j; t_j\rangle$ is a *sub-trajectory* of $T[t_1{:}t_N]$, where $1 \leq i \leq j \leq N$.

A trajectory T has an identifier *tid*, denoted by $T.tid$ and a class label C, denoted by $T.C$. A trajectory and its sub-trajectories are associated with the same identifier and the same class.

We use $T[t]$ to denote the object location at time t, or the interpolated location if t is not in the sampling time list. Let $T(i)$ to denote the i-th sampling point of T, i.e., $T(i) = \langle p_i, t_i\rangle$. We use l_i to denote the i-th segment of trajectory T, i.e., the segment between $T(i)$ and $T(i + 1)$.

The *duration* of trajectory $T[t_i{:}t_j]$, denoted by $|T[t_i{:}t_j]|$, is equal to $t_j - t_i$. Two trajectories T_1 and T_2 are *duration-matched* trajectories if $|T_1| = |T_2|$.

Now we define the similarity of two trajectories which have non-equivalent durations. We first introduce the weight of sampling points, and then define the trajectory distance. In trajectory $T = \langle p_1, t_1\rangle \langle p_2, t_2\rangle \ldots \langle p_N, t_N\rangle$, the *weight* of sampling point p_i, denoted as w_i, is determined by the sampling time of the sampling points which are immediately neighboring to p_i.

$$w_i = \begin{cases} (t_2 - t_1)/2 & i = 1, \\ (t_N - t_{N-1})/2 & i = N, \\ (t_{i+1} - t_{i-1})/2 & otherwise. \end{cases} \tag{6.1}$$

We consider that two trajectories are similar if they have at least one pair similar duration-matched sub-trajectories. We use the mean Euclidean distance to measure the similarity of duration-matched sub-trajectory. Existing time series similarity measures, such as DTW [39], EDR [9] and LCSS [75], finds an optimal match on the *whole* sequences. Since we intend to group the similar sub-trajectories as path patterns, it is not reasonable to compare the similarity on the whole trajectories. Instead, we consider that two trajectories are similar if they have at least one pair similar duration-matched sub-trajectories.

We refer to the mean Euclidean distance between trajectories T_1 and T_2 being valid during the period $[t_0, t_0+\tau]$ [21], which is defined as the definite integral of the Euclidean distance between two moving points during the given period divided by the period time.

$$ED(T_1, T_2) = \frac{1}{\tau}\int_{t_0}^{t_0+\tau} dist(T_1[t], T_2[t])dt \tag{6.2}$$

where $dist(\cdot, \cdot)$ is the Euclidean distance of two points.

By trapezoid rule, Equation (6.2) can be approximated by

$$ED(T_1, T_2) \approx \frac{1}{2\tau}\sum_{k=1}^{n-1}((D(t_k) + D(t_{k+1})) \times (t_{k+1} - t_k)) \tag{6.3}$$

where $D(t_k) = dist(T_1[t_k], T_2[t_k])$ and n is the duration of trajectories. This approximation is guaranteed by an error bound [21].

We extend Equation (6.3) to compute the mean Euclidean distance between two duration-matched sub-trajectories $T_1' = T_1[t_1, t_1 + \tau]$ and $T_2' = T_2[t_2, t_2 + \tau]$ of duration τ as follows.

$$ED(T_1', T_2') = \frac{1}{2\tau}\sum_{k=1}^{n-1}((D'(t_k) + D'(t_{k+1})) \times (t_{k+1} - t_k)) \tag{6.4}$$

where $D'(t_k) = dist(T_1[t_1 + t_k], T_2[t_2 + t_k])$ and n is the duration of trajectories.

To efficiently compute the trajectory distance $ED(T_1[t_1, t_1 + \tau], T_2[t_2, t_2 + \tau])$, we need to interpolate a collection of points in $T_1[t_1, t_1 + \tau]$ and $T_2[t_2, t_2 + \tau]$. Such sampling points are interpolated as follows. Assume $T_1[t_1, t_1 + \tau]$ has raw sampling points $\langle p_1, t_1\rangle$, $\langle p_2, t_1 + t_{k1}\rangle, \ldots, \langle p_m, t_1 + t_{km}\rangle$, and $T_2[t_2, t_2 + \tau]$ has raw sampling points $\langle p_1, t_2\rangle$, $\langle p_2, t_2 + t_{k2}\rangle, \ldots, \langle p_N, t_2 + t_{kn}\rangle$. To compare two duration-matched sub-trajectories, we combine the sampling times of both T_1 and T_2 to be $\mathcal{K} = \{t_{k1}, \ldots, t_{km}\} \cup \{t_{k2}, \ldots, t_{kn}\}$. For $\forall t \in \mathcal{K}$, we interpolate T_1 at time $t_1 + t$ if T_1 does not have a sampling point at this time, and interpolate T_2 at time $t_2 + t$ if T_2 does not have a sampling point at this time. The purpose of interpolation is to make corresponding sampling point pairs on two duration-matched sub-trajectories.

After interpolation, we define the trajectory distance between two duration-matched sub-trajectories as follows.

Definition 16. *(**Trajectory Distance**) Given two duration-matched sub-trajectories $T_1' = T_1[t_1, t_1 + \tau]$ and $T_2' = T_2[t_2, t_2 + \tau]$ of duration τ. The trajectory distance of T_1' and T_2'*

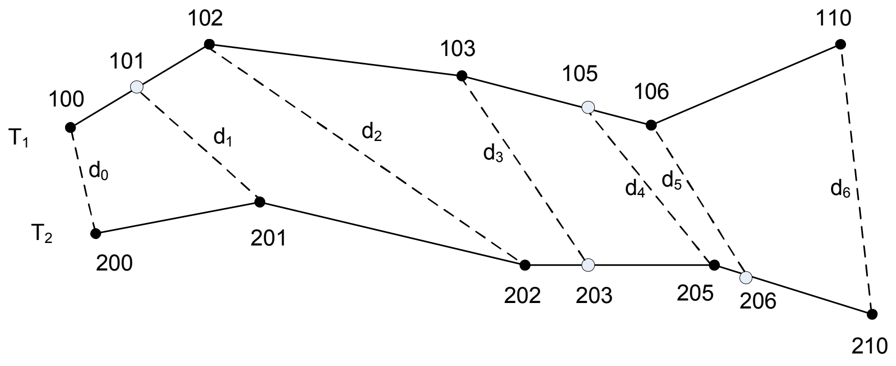


Figure 6.2: Example of trajectory distance computation. Solid points are raw sampling points in trajectories; circle points are interpolated.

is computed as follows.

$$TD(T_1', T_2') = \frac{1}{\tau} \sum_{k \in \mathcal{K}} w_k \times d_k \tag{6.5}$$

where $\mathcal{K}$ is the set of sampling point pairs, w_k is the weight of k-th sampling point pair, d_k is the Euclidean distance of the k-th sampling point pair.

Essentially, trajectory distance is the *weighted* average distance of sampling point pairs, where the weights are measured by the weights of sampling points. Compared to the arithmetic average distance of sampling point pairs, trajectory distance in Definition 16 is more accurate to measure the similarity of duration-matched trajectories. Note that trajectory distance will not be affected by the number of sampling points on trajectories. The trajectory distance of two duration-matched sub-trajectories is determined no matter how many sampling point pairs they have. This property lets trajectory distance be suitable for the similarity computation of compressed trajectories.

For example, Figure 6.2 shows two duration-matched sub-trajectories, T_1[100:110] and T_2[200:210]. After interpolation, each trajectory has seven sampling points. The weight of the sampling points are 0.5, 1, 1, 1.5, 1.5, 2.5, 2, respectively, and the trajectory distance between T_1[100:110] and T_2[200:210] is $\frac{1}{10}(0.5d_0 + d_1 + d_2 + 1.5d_3 + 1.5d_4 + 2.5d_5 + 2d_6)$, which is the mean Euclidean distance over the duration.

6.2 Problem Statement

We define two kinds of features for classification: duration-sensitive region rules and speed-differentiating path rules.

Definition 17. *A duration-sensitive* **region rule** γ *is represented as* $\gamma : R \Rightarrow C$*, in which the antecedent is a discriminative region* R*, denoted by* $\gamma.R$*, and the consequent is the class label* C*, denoted by* $\gamma.C$*.*

Duration-sensitive region rule is different from region cluster [42] because we not only consider the density of the trajectories in the region but also the amount of time the moving objects stay in the region. For this purpose, we define the support of a region R, denoted by $sup(R)$. Let D be the trajectory database and $|D| = \sum_{T \in D} |T|$. $sup(R)$ is the ratio of the sum of trajectory durations within region R over $|D|$. The **support** of region rule $\gamma : R \Rightarrow C$, denoted by $\gamma.sup$, is the ratio of the sum of class C trajectory durations within region R over $|D|$. The **confidence** of region rule $\gamma : R \Rightarrow C$, denoted by $\gamma.conf$, is $\frac{\gamma.sup}{sup(R)}$. Given min_sup and min_conf, we say that a region rule γ is **valid** if $\gamma.sup \geq min\_sup$ and $\gamma.conf \geq min\_conf$.

For valid region rules, we consider that the larger regions are more desirable than the smaller regions because the number of larger regions are usually small, which can improve the classification efficiency.

Speed-differentiating path rule captures the movement and speed of moving objects by utilizing the concept of micro-cluster. A micro-cluster is a group of nearby sampling points of trajectories.

Definition 18. *A* ***path pattern*** $P = (mc_0 \xrightarrow{\alpha_1} mc_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_m} mc_m)$ *is a sequence of micro-clusters with an associated duration interval between consecutive pairs of micro-clusters, where* mc_i *is the i-th micro-cluster and* α_i *is the duration interval from* mc_{i-1} *to* mc_i*.*

Given a path pattern $P = (mc_0 \xrightarrow{\alpha_1} mc_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_m} mc_m)$, if a trajectory T has $m + 1$ consecutive sampling points $p_i, \ldots, p_{i+m}$, such that $p_i \in mc_0$, ..., $p_{i+m} \in mc_m$, and $t_{i+1} - t_i \in \alpha_1$, ..., $t_{i+m} - t_m \in \alpha_m$, we say T supports P or P covers T.

Given a trajectory database D, the **support** of path pattern P, denoted by $sup(P)$, is defined as the ratio of the trajectories in D that support P over the total number of trajectories in D, i.e., $\frac{|\{T|T \in D \wedge T \text{ supports } P\}|}{\# \text{ of trajectories in } D}$. If a trajectory has multiple sub-trajectories which support P, we only count it once. Given min_sup, we say that path pattern P is **frequent** if $sup(P) \geq min\_sup$.

Definition 19. *A speed-differentiating* ***path rule*** γ *is represented as* $\gamma : P \Rightarrow C$, *in which the antecedent is a frequent path pattern* P, *denoted by* $\gamma.P$, *and the consequent is the class label* C, *denoted by* $\gamma.C$.

The **support** of path rule γ, denoted by $\gamma.sup$, is defined as $\frac{|\{T|T \in D \wedge T.C = \gamma.C \wedge T \text{ supports} \gamma.P\}|}{\# \text{ of trajectories in } D}$. The **confidence** of γ, denoted by $\gamma.conf$, is $\frac{\gamma.sup}{sup(P)} = \frac{|\{T|T \in D \wedge T.C = \gamma.C \wedge T \text{ supports } \gamma.P\}|}{|\{T|T \in D \wedge T \text{ supports } \gamma.P\}|}$. Given a path rule γ and a trajectory T, if path pattern $\gamma.P$ covers T and $\gamma.C = T.C$, we say that γ is a covering rule of T.

Given two path rules $\gamma : P \Rightarrow C$, where $P = (mc_0 \xrightarrow{\alpha_1} mc_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_m} mc_m)$, and $\gamma' : P' \Rightarrow C$, where $P' = (mc_0 \xrightarrow{\alpha_1} mc_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_k} mc_k)$, we say that γ' is a **prefix rule** of γ if $k \leq m$. Apparently, $\gamma.sup \leq \gamma'.sup$.

A trajectory may have multiple covering rules. It is important to rank the significance of covering rules for rule selection. In this work, we follow the definition of rule significance proposed in [47, 12].

Definition 20. *A path rule* γ_1 *is more* ***significant*** *than another path rule* γ_2 *if* $(\gamma_1.conf > \gamma_2.conf) \vee (\gamma_1.conf = \gamma_2.conf \wedge \gamma_1.sup > \gamma_2.sup)$.

Besides the Definition 20, given a path rule γ, we consider its prefix rule γ' is more significant for classification, if $(\gamma.sup = \gamma'.sup) \wedge (\gamma.conf = \gamma'.conf)$. This is because the path patterns in prefix rules are shorter, which can improve the efficiency of path pattern mining and trajectory classification.

Definition 21. *Given a trajectory database D and min_sup, the* **top-k covering path rules** *of trajectory T are the top-k most significant path rules out of all covering rules of T, and their supports are not less than min_sup.*

Problem Definition. Given a training trajectory database D and a testing trajectory database $\mathcal{P}$, a minimal support min_sup, a minimal confidence min_conf and the number k, our goals are 1) mine the valid region rules from D, 2) mine the top-k covering rules from D, and 3) select the region rules and path rules to build the classifiers and predict the class label of test trajectories.

6.3 Solution Overview

Our solution is divided into three phases, as shown in Figure 6.3. For easy presentation, Figure 6.3 only shows two classes of trajectories which are recognized by solid and dashed lines.

In the first phase, we partition the regions based on the trajectory distribution in a top-down space partition manner. A region is partitioned into two regions if those two regions are capable to better discriminate trajectory classes than one region. The partition evaluation criteria could be information gains [57], fisher score [15] or others.

In the second phase, we summarize the trajectory database into a simplified trajectory network by a bottom-up points clustering approach. For the efficiency concern, the points clustering performs only on a subset of trajectory database which are out of the spatial range of valid region rules, because the trajectories in the region rules are covered by region rules.

In the last phase, we mine the valid region rules and top-k covering path rules for

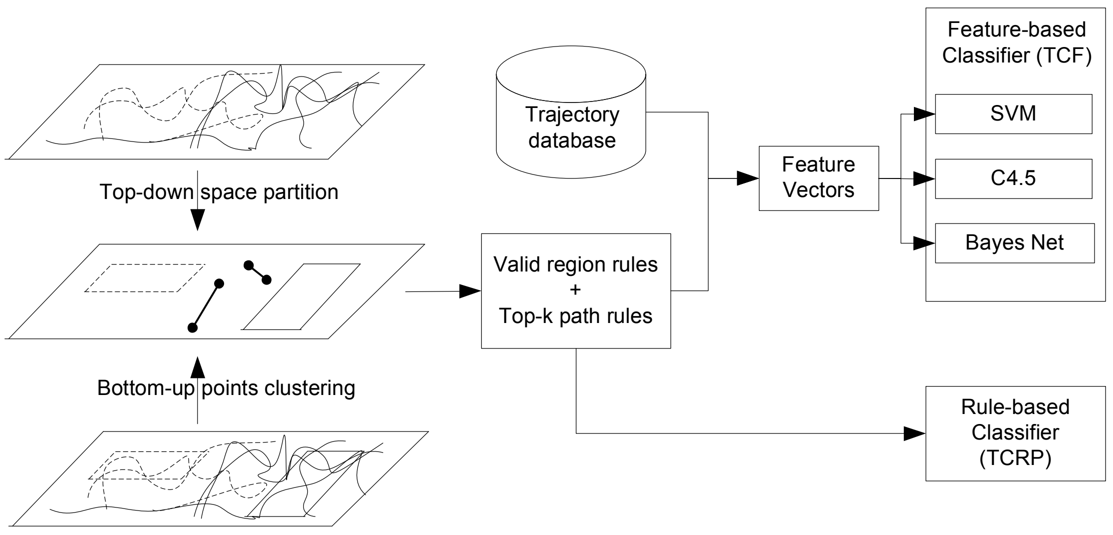

Top-down space partition
Bottom-up points clustering
Trajectory database
Valid region rules + Top-k path rules
Feature Vectors
Feature-based Classifier (TCF)
SVM
C4.5
Bayes Net
Rule-based Classifier (TCRP)

Figure 6.3: Our solution overview

classification. We build two classifiers based on two classification strategies. This first classifier, named TCF (Trajectory Classification based on Features), transforms the trajectory network into feature vectors based on the valid region rules and top-*k* covering path rules and feeds into SVM, C4.5 and Bayes Net to predict the class labels of test trajectories. Hence, TCF has three versions based on SVM, C4.5 and Bayes net, respectively. The second classifier, named TCRP (Trajectory Classification based on Region and Path rules), is a rule-based classifier. TCRP selects the top-*k* rules that have the highest scores with respect to the test trajectory and votes the class label to be the one with the majority vote.

6.4 Region Rules

In [42], the region features are the region-based clusters, presented by the homogeneous rectangular regions where one major class has at least ϕ trajectories and the other classes do not. They select an optimal partition line to partition the spatial plane each time. However, this partition method may miss some local features [19] because the partition line partitions the whole spatial plane in each iteration. Figure 6.4(a) illustrates an example, where the whole space is partitioned into six regions by three partition lines. For this trajectory distribution, we expect a better partition pattern shown in Figure 6.4(b), where the whole space is partitioned into four regions .

In [42], discriminative regions are found where one class of trajectories dominates the other classes regardless of how much time the trajectories remain within the regions. In addition, the regions are found by selecting an optimal partition line to partition the *whole* spatial plane. This may results in some locally discriminative regions being missed. For example, Figure 6.4 shows four trajectories of two classes which are recognized by solid and dashed lines, respectively. In Figure 6.4(a), the whole spatial plane is partitioned into six regions by three partition lines. For this trajectory distribution, a better partition pattern is shown in Figure 6.4(b), where the whole space is partitioned into four regions.

Targeting on the two limitations in [42], we incorporate the duration information of trajectories in deriving the discriminative regions. We also employ a scheme to partition the space in a divide-and-conquer manner utilizing information gain to detect the locally discriminative regions.

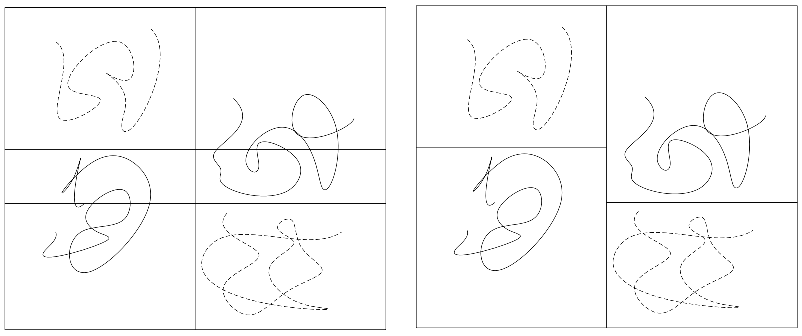

(a) Partition the whole spatial plane (b) Partition the regions

Figure 6.4: An example to show the different results of two region partition approaches

The incorporation of duration information of trajectories is achieved as follows: A trajectory exerts an influence on a region, which is measured by the duration time the trajectory stays within the region. This makes sense because a longer time duration of one class trajectory within a region indicates a higher influence of that class trajectory to the region. The influence of trajectory class C to region R, denoted by $inf(R)_C$, is the accumulated influence from all trajectories T in training trajectory database, where $T.C = C$.

A grid structure has been employed to make the evaluation of the influences of trajectory class on a region computationally feasible. Each line segment l exerts a degree of influence, which is measure by the length of l, to its nearby cells on the grid. Assume n cells are influenced by l, so the line segment l exerts $\frac{|l|}{n}$ influence to each one of its nearby n cell. The influence of a cell is the sum of influence from all line segment in trajectory database. A region R consists of a set of cells, so the influence of R is the sum of influence on these cells.

Let $\mathcal{C}$ be the set of class labels in trajectory database. The support of region R is sum of region influence from all trajectory classes, i.e., $\sum_{C\in\mathcal{C}} inf(R)_C$. Assume we have a region R and a partition line $X = x$ such that R is partitioned into two regions R^- and R^+. Let $P(C)$ be the influence weight of class C on region R; $P(x^-)$ and $P(x^+)$ be the support of region R^- and R^+, respectively; $P(C|x^-)$ and $P(C|x^+)$ be the influence weight of class C on region R^- and R^+, respectively. The information gain of the partition line $X = x$ is measured as

$$\begin{aligned} IG(C|X = x) = H(C) - H(C|X = x) = &-\sum_{C\in\mathcal{C}} P(C)\log P(C) \\ &+ \sum_{x\in\{x^-,x^+\}} P(x) \sum_{C\in\mathcal{C}} P(C|x)\log P(C|x) \end{aligned} \tag{6.6}$$

A positive information gain implies that the partition line $X = x$ distinguishes the trajectory class on region R, and negative information gain implies that $X = x$ is not discriminative. Partition line $X = x_1$ is more discriminative than line $X = x_2$, if

$IG(C|X = x_1)$ is greater than $IG(C|X = x_2)$. The selection of partition line on Y axis is similar to that on X axis.

We design a space partition tree to facilitate the partition process. In a space partition tree, the root node corresponds to the whole space. A non-leaf node is marked by "|" or "−" if its corresponding region is partitioned by a partition line on X or Y axis, respectively. Each leaf node corresponds to a region which will not be further partitioned. Figure 6.5 gives an example of space partition tree and the corresponding region of each node.

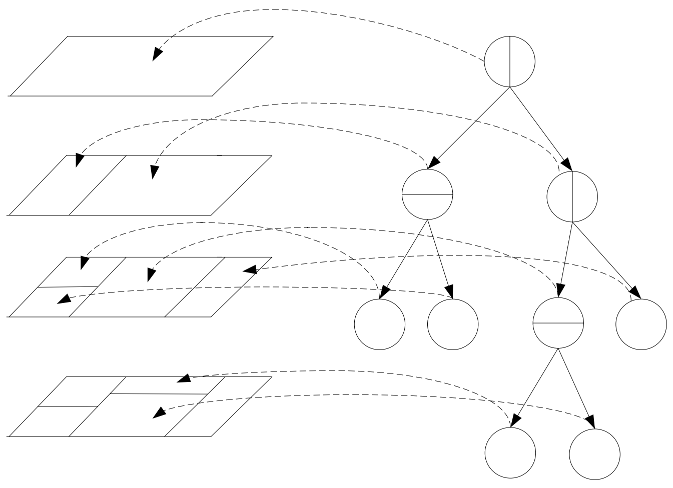


Figure 6.5: An example of space partition tree

Algorithm 13 presents a method to partition regions with the facilitation of a stack structure. Line 1 detects the trajectory density on grid G. Lines 2-4 initialize a stack S and a candidate region R of maximal area according to the scope of grid G, and push R into stack S. In Lines 5-14, we evaluate the candidate regions in stack S as follows. Line 6 pops the top element in S. If R is a valid region, whose support is no less than min_sup and confidence is no less than min_conf, we translate R into a region rule γ based on the dominant class on R, and insert γ into the region rule set RRS (Lines 7-10). Otherwise, Line 12 employs a sweep line moves on X and Y axis on R to search the optimal partition line pl. The goodness of partition is measured by information gain as

Algorithm 13: Region_Partition

```
input  : A trajectory database $\mathcal{D}$;
         A grid $G$;
         Minimal support $min\_sup$;
         Minimal confidence $min\_conf$.
output: The region rule set $RRS$.
1  Density estimation of $\mathcal{D}$ on $G$;
2  Initialize a stack $S$;
3  $R \leftarrow G$;
4  Push($R$, $S$);
5  while $S \neq \emptyset$ do
6      $R$ = Pop($S$);
7      if ($R.sup \geq min\_sup$) then
8          if $R.conf \geq min\_conf$ then
9              Translate $R$ into region rule $\gamma$;
10             Insert $\gamma$ into $RRS$;
11         else
12             Select the optimal partition line $pl$;
13             Partition $R$ into regions $R^-$ and $R^+$;
14             Push($R^-$, $S$);
15             Push($R^+$, $S$);
16 return $RRS$;
```

shown in Equation 6.6, and the step width of sweep line is the cell side. Based on *pl*, Line 12 partitions R into two non-overlapping regions R^- and R^+ along *pl*. Lines 14-15 push the two regions R^- and R^+ into stack S. Line 16 returns the region rules.

Region rules are the simple but useful features for classification. However, only region rules are not enough for classification because the numbers of discovered region rules are quite small or even zero for some datasets whose trajectory distribution of different classes are highly mixed, like the trajectories in the urban transportation. This motivates us to define and find more complicated features for classification.

6.5 Path rules

Our proposed path rule mining method consists of three phases. First, we summarize the trajectory data which cannot be covered by region rules, into a network structure named trajectory network, where vertices are the Regions-of-Interest and edges are the movement paths with duration information. Second, in order to discover the discriminative features for trajectory classification, we mine the top-*k* covering path rules based on the trajectory network. Finally, we build two classifiers based on the top-*k* covering path rules, respectively.

In this section, we first introduce a trajectory network to model the trajectory database, and then introduce a path pattern tree to facilitate the discovery of path rules. Section 6.6 will give the details to build classifiers.

6.5.1 Trajectory Network

Definition 22. *A* **trajectory network** *M is a directed graph which can be represented as $M = \langle \mathcal{V}, \mathcal{E} \rangle$, where $\mathcal{V}$ is the set of vertices, $\mathcal{E}$ is the set of edges.*

In trajectory network, each vertex is a group of nearby sampling points of trajectories, and each edge is a group of temporally close segments of trajectories. Each vertex pair may have one or multiple edges to indicate the movement speeds between them. The definitions of vertices and edges are the extension of the cluster feature vector in [89] by considering the class labels of sampling points and segments, respectively.

Let the sampling point p_t of trajectory T at time t be represented by $(\vec{p_t}, w_t)$, where $\vec{p_t}$ is the position and w_t is the weight as defined in Equation (6.1). The class of p_t is $T.C$.

Definition 23. *Each* **vertex** *v in a trajectory network $M.\mathcal{V}$ is labelled as a tuple $(N, \overrightarrow{W}, \overrightarrow{S}, SS)$, where N is the number of sampling points in v, $\overrightarrow{W}$ is a vector of weight for the* *classes in v, $\overrightarrow{S}$ is the weighted sum of the N data point locations, i.e., $\sum_{i=1}^{N} w_i \vec{p_i}$, and SS is the square sum of the N data point locations, i.e., $\sum_{i=1}^{N} \vec{p_i}^2$.*

Given a vertex $v = (N, \overrightarrow{W}, \overrightarrow{S}, SS)$, and let $|C|$ be the number of classes in v. We also derive its *weight* (W), *entropy* (H), *centroid*($\overrightarrow{O}$) and *radius*(R) as follows.

$$W(v) = \sum_{i=1}^{|C|} \overrightarrow{W}_i \tag{6.7}$$

$$H(v) = -\sum_{i=1}^{|C|} \frac{\overrightarrow{W}_i}{W} \log \frac{\vec{W}_i}{W} \tag{6.8}$$

$$\overrightarrow{O}(v) = \frac{\overrightarrow{S}}{W} \tag{6.9}$$

$$R(v) = \frac{\sum_{i=1}^{N} (\vec{p_i} - \overrightarrow{O})^2}{N} \tag{6.10}$$

The i-th segment l_i of trajectory T is associated with two vertices v_s and v_e, where v_s (v_e) includes the starting (ending) sampling point of l_i. The weight of l_i is equal to 1. The class of l_i is the $T.class$. An edge of trajectory network is the summary of a set of directed segments.

Definition 24. *In trajectory network M, an* **edge** *e between two vertices v_s and v_e is labelled as a tuple (N, $\overrightarrow{W}$, DT, v_s, v_e), where N is the number of segments, $\overrightarrow{W}$ is a vector of segment number for all classes in e, DT is the sum of duration time of all segments in e, v_s is the starting vertex, v_e is the ending vertex.*

Given an edge $e = (N, \overrightarrow{W}, DT, v_s, v_e)$ and let $|C|$ be the number of classes in e. We can derive its *weight* (W), *entropy* (H), and *average duration*(D) as follows.

$$W(e) = N \tag{6.11}$$

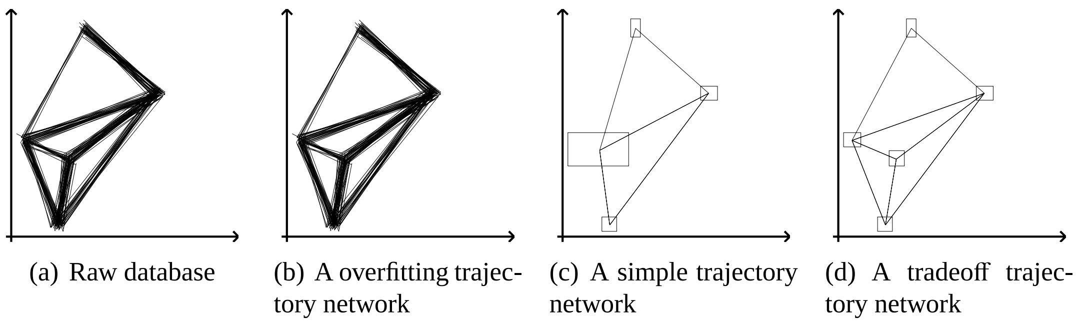

(a) Raw database (b) A overfitting trajectory network (c) A simple trajectory network (d) A tradeoff trajectory network

Figure 6.6: Trajectory network selection

$$H(e) = -\sum_{i=1}^{|C|} \frac{\overrightarrow{W_i}}{W} \log \frac{\overrightarrow{W_i}}{W} \tag{6.12}$$

$$D(e) = \frac{DT}{W} \tag{6.13}$$

Note that there can be many trajectory networks for a given trajectory database. Given a trajectory database in Figure 6.6(a), a trajectory network which contain too many vertices and edges, as shown in Figure 6.6(b), is not a good one for deriving path rules due to overfitting. On the other hand, the trajectory network which contain too few number of vertices and edges, as shown in Figure 6.6(c), is not good either because of the large information loss. Between these two extreme cases, a good trajectory network is shown in Figure 6.6(d), which is a reasonable tradeoff.

We propose three criteria to evaluate the goodness of a trajectory network. First, the trajectory network should contain the least possible number of vertices and edges. Second, the trajectory network should minimize the amount of information loss when it models the trajectory database, that is, the network should minimize the total distance from all trajectories in the trajectory database to the network. Third, the trajectory network should increase the discriminative power for classification. Each vertex and each edge are expected to be discriminative, i.e., low entropy in vertices and edges.

Based on these three criteria, we define a Minimal Description Length (MDL) cost to find a good trajectory network. The MDL cost consists of two components, network codelength $L(M)$ and data codelength $L(D|M)$ [27]. Network codelength $L(M)$ is the length, in bits, of the description of the candidate trajectory network.

Definition 25. *(**Network Codelength**) Let $M = \langle \mathcal{V}, \mathcal{E} \rangle$ be a network with $|\mathcal{V}|$ vertices and $|\mathcal{E}|$ edges. The information of M can be transmitted using one bit per vertex and one bit per edge. Therefore, the code length is*

$$L(M) = |\mathcal{V}| + |\mathcal{E}| \tag{6.14}$$

The data codelength $L(D|M)$ is the length, in bits, of the description of the data when encoded with the help of the trajectory network. $L(D|M)$ is the sum of the entropies in vertices and edges, and the information of weighted trajectory distance, in bits, from trajectory database D to trajectory network M.

Definition 26. *(**Data Codelength**) Given a trajectory network $M = \langle \mathcal{V}, \mathcal{E} \rangle$ for a trajectory database D, the data description information consists of vertex entropy, edge entropy and the information of weighted trajectory distance.*

$$L(D|M) = \sum_{i=1}^{|\mathcal{V}|} H(v_i) + \sum_{i=1}^{|\mathcal{E}|} H(e_i) + C(D, M) \tag{6.15}$$

where $H(v_i)$ and $H(e_i)$ are the vertex entropy of vertex v_i and edge entropy of edge e_i, respectively, and $C(D, M)$ is the information of weighted trajectory distance from D to M .

To compute $C(D, M)$, we first introduce the distance from D to M, denote by $TD(D, M)$. $TD(D, M)$ is the weighted trajectory distance of each trajectory T to its most similar path P on the trajectory network M.

$$TD(D, M) = \frac{1}{|D|} \sum_{T \in D, P \in M} |T| \times TD(T, P) \tag{6.16}$$

where $|D| = \sum_{T \in D} |T|$.

Intuitively, a small $TD(D, M)$ value indicates a small weighted trajectory distance from the trajectory database D to the trajectory network M. We employ $C(D, M)$, in terms of bits, to measure $TD(D, M)$ as follows.

$$C(D, M) = -\sum_{T \in D} \frac{|T|}{|D|} \log_2 g(TD(T, P)) \tag{6.17}$$

where $g(TD(T, P))$ is a underlying distribution of $TD(T, P)$.

We can use the normal distribution or exponential distribution to model the underlying distribution of $TD(T, P)$. Assuming that $TD(T, P)$ follows the normal distribution $\mathcal{N}(0, \sigma^2)$, where σ is the standard deviation.

Note that σ can be assigned as a parameter or be estimated to be the standard deviation of sampling point positions. In kernel density estimation [62], a global bandwidth is assigned to all data points on the spatial plane,

$$h^{(n)} = 1.06 \cdot \sigma^{(n)} \cdot n^{-\frac{1}{5}} \tag{6.18}$$

where $\sigma^{(n)}$ is the standard deviation distance of n elements. The global bandwidth of Equation (6.18) provides an estimation of σ in Equation (6.17). A further empirical study shows that our classifier obtains the highest accuracy when the σ value is selected to be around the global bandwidth $h^{(n)}$.

The best trajectory network to model the distribution of trajectory database D is the one that minimizes the MDL cost $L(M) + L(D|M)$ [27].

Algorithm TrajNet

The computation of the global minimum description length is quite costly, and exact approaches requires time and space complexity that increases exponentially to the input size. Here, we adopt an approximate approach that utilize a forward search of local optimal solution.

The basic idea of our approximate algorithm is to select the local optimal trajectory network based on the current network M. A candidate network model M' is obtained by merging two vertices or two edges of M. We define the **MDL gain** as the MDL difference of M and M' as follows.

$$MDL\_gain = MDL(M) - MDL(M') \tag{6.19}$$

A large MDL gain indicates a large decrease of network codelength and a small increase of data codelength. The candidate network model which obtains the maximal MDL gain is the local optimal. Similar to the agglomerative hierarchical clustering [28], our approximate algorithm, called TrajNet, selects the local optimal network in each iteration. TrajNet has three main steps.

1. **Initialization.** The trajectory database is treated as an initial trajectory network, where each vertex is a sampling point and each edge is a segment;
2. **Merge Vertices.** We fix the edges in the initial trajectory network, and iteratively merge two vertices which always cause the positive largest MDL_gain value;
3. **Merge Edges.** We fix the vertices in the current trajectory network, and for each vertex pair, we iteratively merge two edges which cause the positive largest MDL_gain values.

In the phase of merging vertices, as there is no change in the number of edges in the first phase, each trajectory always has a duration matched path in trajectory network. In this phase, the overall distance from trajectory database D to a trajectory network M can be computed as follows.

Lemma 10. *The overall distance from trajectory database D to a trajectory network M is the weighted vertex radius summation, i.e., $\sum_{i=1}^{|\mathcal{V}|} W_i \times R_i$, where W_i is the weight of the i-th vertex and R_i is the radius of i-th vertex.*

Proof: For each trajectory T, there is a duration matched path P in trajectory network for T. Assume that T has N sampling points and P has N vertices. From Definition 16, $TD(T,P)$=$\sum_{k=1}^{N} w_k \times d_k$, where w_k is the weight of k-th sampling point and d_k is the distance between k-th sampling point and k-th vertex. By combing the distances from all trajectories to the trajectory network, we have

$$TD(D,M) = \frac{1}{|D|}\sum_{T\in D}|T| \times TD(T,P) = \frac{1}{|D|}\sum_{T\in D}\sum_{k=1}^{N} w_k \times d_k$$
$$= \frac{1}{|D|}\sum_{i=1}^{\mathcal{V}}\sum_{j=1}^{|v_i|} w_{ij} \times d_{ij} = \sum_{i=1}^{\mathcal{V}}\sum_{j=1}^{|v_i|}\frac{w_{ij}}{|D|} \times d_{ij} = \sum_{i=1}^{\mathcal{V}} W_i \times R_i$$

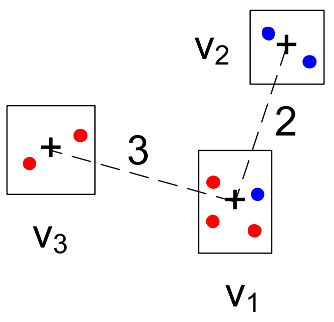
□

Lemma 10 states that the distance of trajectory database and trajectory network is determined by the weighted vertex radius. By merging two vertices v_i and v_j into a larger vertex v, the MDL gain is

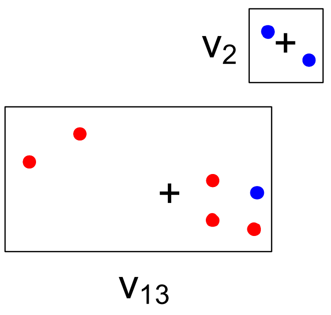
$$\begin{aligned} & 1 + H(v_i) + H(v_j) - H(v) + C(v_i) + C(v_j) - C(v) \\ = \;& 1 + H(v_i) + H(v_j) - H(v) + c_v(W_i R_i^2 + W_j R_j^2 - W R^2) \end{aligned}$$

where $W = W_i + W_j$ and c_v is a constant coefficient to smooth the distance error of vertices. Here, $c_v = 1/(2\sigma^2 w log_e 2)$, w is the arithmetic average weight of all sampling points.

For example, Figure 6.7(a) and Figure 6.7(b) show the process to merge vertices v_1, v_2 and v_3. We consider the three cases. 1) Merge v_1 and v_2 to be a new vertex v_{12} which causes MDL_gain = 0.68 bits; 2) Merge v_1 and v_3 to be a new vertex v_{13} which



(a) Before the merge



(b) Merge v_1 and v_3

Figure 6.7: An example of vertex merge. The "Red" and "Blue" colors indicate the different classes, and the "+" label indicates the centroid of each vertex.

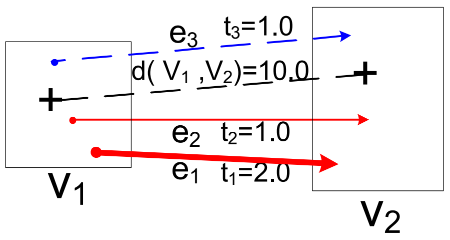
which causes MDL_gain = 0.93 bits; 3) Merge v_2 and v_3 to be a new vertex v_{23} which causes MDL_gain = -0.23 bits. Appendix 7.2.1 gives the computation details of the three cases. Since Case 2 leads to a largest MDL_gain, we select to merge v_1 and v_3 to be a new vertex v_{13}, as shown in Figure 6.7(b).

In the phase of merging edges, we update the number of edges and the distance error after merging two edges. Assume that we merge two edges $e_i = v_1 \xrightarrow{t_i} v_2$ and $e_j = v_1 \xrightarrow{t_j} v_2$. After merging, the weighted duration is $t = \frac{w_i t_i + w_j t_j}{w_i + w_j}$, so the Euclidean distance errors are $d_i = |t - t_i| \times d(v_1, v_2)$ and $d_j = |t - t_j| \times d(v_1, v_2)$, respectively. Again, we use normal distribution to model the Euclidean distance errors. In terms of bits, such error information can be transmitted by $-w_i \log_2 \frac{1}{\sqrt{2\pi}\sigma}(exp(-\frac{d_i^2}{2\sigma^2})) - w_j \log_2 \frac{1}{\sqrt{2\pi}\sigma}(exp(-\frac{d_j^2}{2\sigma^2}))$ bits. By merging e_i and e_j into a larger edge e, the MDL gain is

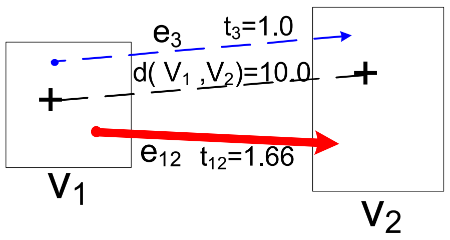
$$\begin{aligned} & 1 + H(e_i) + H(e_j) - H(e) + C(e_i) + C(e_j) - C(e) \\ = \; & 1 + H(e_i) + H(e_j) - H(e) - c_e(w_i d_i^2 + w_j d_j^2) \end{aligned}$$

where c_e is a constant coefficient to smooth the distance error of edges. Here, $c_e = 1/(2\sigma^2 log_e 2)$.

For example, Figure 6.8 shows the process to merge edges. Assume that we have three edges e_1, e_2 and e_3 that move from vertex v_1 to vertex v_2. We consider three cases.



(a) Before the merge



(b) Merge e_1 and e_2

Figure 6.8: An example of merge edge. The "Red" and "Blue" colors indicate the different classes, and the "+" label indicates the centroid of each vertex.

1) Merge e_1 and e_2 to be a new edge e_{12}, which causes MDL_gain = 0.52 bits; 2) Merge e_2 and e_3 to be a new edge e_{23}, which causes MDL_gain = 0.0 bits; 3) Merge e_1 and e_3 to be a new edge e_{13}, which causes MDL_gain = -0.48 bits. Appendix 7.2.2 gives the computation process of the three cases. Since Case 1 leads to a largest MDL_gain, we select to merge e_1 and e_2 to be a new edge e_{12}, as shown in Figure 6.8(b).

Algorithm 14 gives the psuedocode of TrajNet algorithm. The trajectory database is treated as an initial trajectory network, where each vertex is a sampling point and each edge is a segment (Lines 1-2). This initial trajectory network has the maximal codelength value and the minimal data codelength. The rest of algorithm is divided into two phases. In the first phase, we call *MergeVertex*() to merge vertices (Line 3). *MergeVertex*() fixes the edges in the current trajectory network, and successively merge two vertices which always cause the largest decrease of MDL value. The merging of two vertices will decrease the number of vertices by 1, update the vertex entropy and enlarge the vertex radius. In the second phase, we fix the vertices in trajectory network, and for each vertex pair, we call *MergeEdge*() merge two edges which cause the largest decrease of MDL value (Lines 4-5). Line 6 returns the result.

Algorithm 15 shows the merging of vertices. Line 1 calls the function *NN_search*() to find the nearest neighbor of vertex x, which can cause the largest MDL_gain after merging with x. Lines 3-4 select the vertex which has the minimal nearest neighbor distance and merges it with its nearest neighbor to obtain a new vertex w, whose sam-

Algorithm 14: TrajNet

```
input  : A trajectory database $D$
output: Trajectory network $M$
1 $M.\mathcal{V} \leftarrow$ Sampling points of $D$;
2 $M.\mathcal{E} \leftarrow$ Segments of $D$;
3 MergeVertex($M.\mathcal{V}$);
4 foreach vertex pair $(v_s, v_e)$ in $M.\mathcal{V}$ do
5     MergeEdge($M.\mathcal{E}, v_s, v_e$);
6 return $M$.
```

pling points are the union of the nearest vertex pair. Line 5 computes the value of MDL_gain. If MDL_gain is greater than 0, Lines 7-8 attach the edges of two vertices to w, and remove two vertices from trajectory network. Line 9 finds the nearest neighbor of w. Lines 10-16 update the nearest neighbor distances for the other vertices whose nearest neighbors are affected by the vertex merging operation. Line 17 inserts the new vertex w into the vertex set of trajectory network.

Similar to Algorithm 15, *MergeEdge*() initializes segments to be small units for clustering. Algorithm 16 shows the iteration of merging edges whose starting vertex is v_s and ending vertex is v_e. Line 1 calls *NN_search*() to find the nearest neighbor of edge x, which can cause the largest MDL_gain after merging with x. Line 3-4 select the edge which has the minimal nearest neighbor distance and merges it with its nearest neighbor to obtain a new edge w. After edge merging, the starting and ending vertex of w are v_s and v_e respectively, and the duration of w is the weighted duration of u and v. Line 5 computes the value of MDL_gain. If MDL_gain is greater than 0, Lines 7 removes u and v from the edge set of trajectory network. Line 8 finds the nearest neighbor of w in all edges whose starting vertex is v_s and ending vertex is v_e. Lines 9-15 update the nearest neighbor distances for the other edges whose nearest neighbors are affected by the edge merging operation. Line 16 inserts the new edge w into the edge set.

Theorem 5. *The time complexity of Algorithm 14 is $O(n^2 + m^2)$ if no spatial index is*

Algorithm 15: MergeVertex($\mathcal{V}$)

```
foreach $x \in \mathcal{V}$ do $x.closest \leftarrow NN\_search(x, \mathcal{V})$;
while true do
    $u \leftarrow extract\_min(\mathcal{V})$; $v \leftarrow u.closest$;
    $w \leftarrow merge(u, v)$;
    if MDL_gain has no increase after merge then
        break;
    $attach\_edges(u, w, \mathcal{E})$; $attach\_edges(v, w, \mathcal{E})$;
    $delete\_vertex(u, \mathcal{V})$; $delete\_vertex(v, \mathcal{V})$;
    $w.closest \leftarrow NN\_search(w, \mathcal{V})$;
    foreach $x \in \mathcal{V}$ do
        if x.closest is either u or v then
            if $dist(x, x.closest) \leq dist(x, w)$ then
                $x.closest \leftarrow NN\_search(x, \mathcal{V})$;
            else $x.closest \leftarrow w$;
        else if $dist(x, x.closest) \geq dist(x, w)$ then
            $x.closest \leftarrow w$;
    $insert\_vertex(w, \mathcal{V})$;
```

utilized, where n and m are the vertex number and edge number in the initial trajectory network, respectively.

Proof: The time complexity of *MergeVertex*() is $O(n^2)$ if no spatial index is utilized, on account of nearest neighbor search. In Line 4 of Algorithm 14, there are n^2 vertex pairs. We analyze the time complexity of merging edges by the average case and the worst case. On average, each vertex pair has $\frac{m}{n^2}$ edges, the time complexity of *MergeEdge*() is $O((\frac{m}{n^2})^2)$ and the time complexity of merging edges is $O((\frac{m}{n})^2)$. In worst case, one vertex pair is associated with all m edges and the time complexity of merging edges is $O(m^2)$. In summary, for Algorithm 14, its average time complexity is $O(n^2 + (\frac{m}{n})^2)$, and its worst time complexity is $O(n^2 + m^2)$.

□

Algorithm 16: MergeEdge($\mathcal{E}$, v_s, v_e)

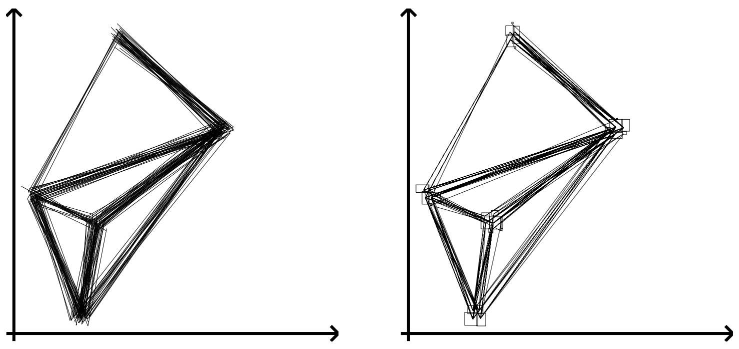
```
foreach $x \in \mathcal{E} \wedge x \in (v_s, v_e)$ do $x.closest \leftarrow NN\_search(x, \mathcal{E}, v_s, v_e)$;
while $true$ do
    $u \leftarrow extract\_min(\mathcal{E}, v_s, v_e)$; $v \leftarrow u.closest$;
    $w \leftarrow merge(u, v)$;
    if $MDL\_gain$ has no increase after merge then
        break;
    $delete\_edge(\mathcal{E}, u)$; $delete\_edge(\mathcal{E}, v)$;
    $w.closest \leftarrow NN\_search(w, \mathcal{E}, v_s, v_e)$;
    foreach $x \in \mathcal{E}$ do
        if $x.closest$ is either $u$ or $v$ then
            if $dist(x, x.closest) \leq dist(x, w)$ then
                $x.closest \leftarrow NN\_search(x, \mathcal{E}, v_s, v_e)$;
            else $x.closest \leftarrow w$;
        else if $dist(x, x.closest) \geq dist(x, w)$ then
            $x.closest \leftarrow w$;
    $insert\_edge(w, \mathcal{E}, v_s, v_e)$;
```

Approximate Trajectory Network Initialization

The time complexity of Algorithm 14 can be high because of the large number of sampling points n and the large number of segments m in the initial trajectory network which is obtained by Lines 1-2 of TrajNet Algorithm. We call it *data-based initialization*.

To avoid the high time complexity of data-based initialization, we adopt a *grid-based initialization*: By imposing a grid on the spatial plane, the sampling points in one cell are initialized to be a vertex. The grid-based initialization is more efficient because the number of vertices $\hat{n}$ is far less than the number of sampling points n, $\hat{n} \ll n$. For example, Figure 6.9 shows the initial trajectory networks based on the two initialization strategy, where the initial trajectory network in Figure 6.9(b) has less vertices than that in Figure 6.9(a). With a proper control of grid granularity, the grid-based initialization leads to the quite similar or the same trajectory network to the trajectory network obtained by data-based initialization.

Here, we introduce parameter c to control the grid granularity. A smaller value of c



(a) Data-based initialization (b) Grid-based initialization

Figure 6.9: Initial trajectory networks

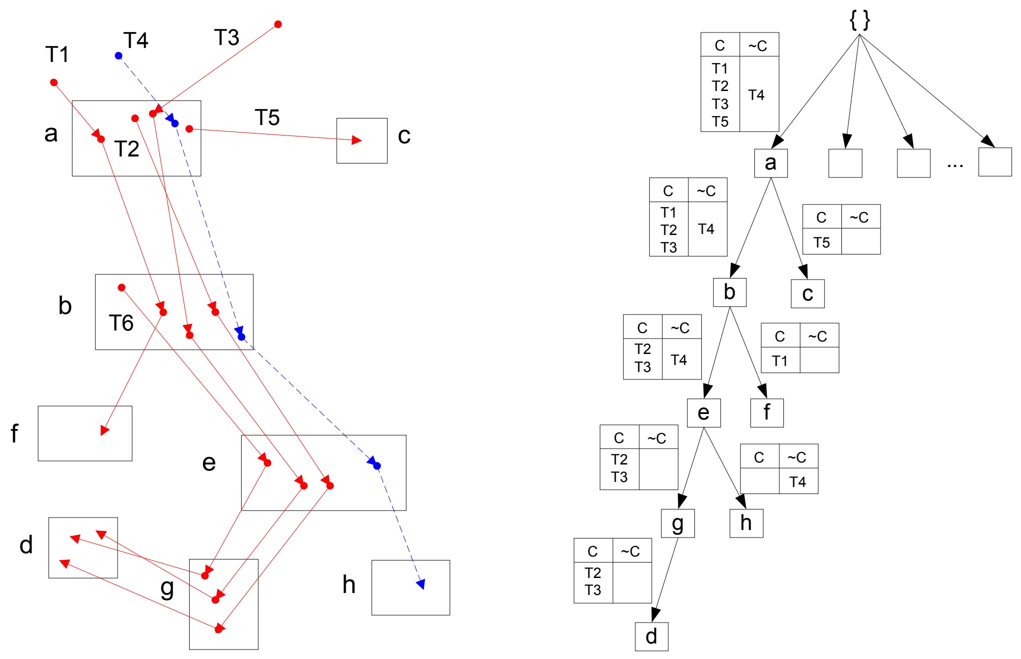
results in a more precise initial trajectory network and a longer clustering time. A larger value of c results in a coarser initial trajectory network and a shorter clustering time. Note c is not supposed to be larger than the expected micro-cluster side σ; Otherwise, the initial trajectory network will depart far from the optimal trajectory network. Our empirical studies show that we obtain a good trajectory network when c is about half of σ.

Similarly, we group the segments which are associated with two vertices and have close durations, to be an initial edge. This also leads to smaller number of edges $\hat{m}$ in the initial trajectory network, where $\hat{m} \ll m$. In summary, the time complexity of for the grid-based initialization is $O(\hat{n}^2 + \hat{m}^2)$.

6.5.2 Path Pattern Tree

To generate speed differentiating path rules, we utilize a path pattern tree which enumerates all possible paths in the trajectory network. The root node of path pattern tree is an empty set, and each vertex in trajectory network is a level one child node in the tree.

In the trajectory network, each edge n_e maintains a set of trajectory ids to indicate the trajectories which move along this edge. Similarly, its corresponding edge p_e in the



(a) Trajectory network (b) Path pattern tree

Figure 6.10: An example of trajectory network and its path pattern tree. Red solid trajectories are class C, and blue dashed trajectories are class $\neg C$.

path pattern tree also maintains a projected table which stores the class distribution of trajectory ids. The projected table of p_e is obtained by intersecting the projected table of p_e's parent with the trajectory ids in n_e. Clearly, the trajectory ids in the projected table is a subset of the trajectory ids in n_e because p_e may have the prefixes in path pattern tree.

For example, Figure 6.10(a) shows a trajectory network with seven vertices and the associated segments. Based on this trajectory network, we build the path pattern tree as shown in Figure 6.10(b). The edge from root to a is associated with a projected table which stores the trajectory ids in vertex a. The edge from node a to b is associated with a projected table which contains three trajectories $T1$, $T2$ and $T3$ of class C and one trajectory $T4$ of class $\neg C$. The edge from node b to e is associated with a smaller projected table which contains $T2$, $T3$ and $T4$. Note that $T6$ is not included in this projected table because it does not start from node a.

From Definition 20, support and confidence are two measures of rule significance. In a path pattern tree, each node can generate several path rules according to the object ids and their class distribution in the associated projected table. We can estimate the upper bound of the significance of all path rules which are generated from the projected table.

Lemma 11. *The upper bound of significance pair (confidence, support) for class C in projected table is*

$$\begin{cases} (\frac{sup(C)}{min\_sup}, min\_sup) & if\ sup(C) < min\_sup, \\ (1.0, sup(C)) & otherwise. \end{cases} \tag{6.20}$$

where $sup(C)$ is the number of object ids in column C of projected table over the total number of trajectories in D.

Proof: From Definition 20, we need to guarantee that the support of path rule is not less than min_sup. Let $|D|$ to be the number of trajectories in database D. There are two cases. 1) If $sup(C) < min\_sup$, the projected table will contain at least $(min\_sup - sup(C)) \times |D|$ object ids of the classes other than class C, in order to the overall support of projected table is no less than min_sup. In this case, the maximal confidence of path rule for class C is $\frac{sup(C)}{min\_sup}$. 2) If $sup(C) \geq min\_sup$, the most significant path rule will be obtained if the projected table only contains the object ids of class C. In this case, the confidence reaches the maximal value 1.0 and the support is $sup(C)$. □

6.5.3 Top-k Covering Path Rule Set

In this section, we present the algorithm to mine top-k covering path rules for classification. For a trajectory database D, the top-k covering path rules set is the combination of top-k covering path rules of each trajectory $T \in D$ after removing the duplicate path rules.

Note that we do not mine the valid path rules, whose confidences are greater than min_conf, to build classifiers because the confidence threshold cannot control the number of valid path rules. Even with the high confidence threshold, a large number of valid path rules are generated in the datasets of long trajectories, which results in the inefficient to build classifiers and predict the class labels of test trajectories. In contrast, the choice of k is semantically clear, and we can easily control the number of top-k covering path rules to build classifiers.

Based on the path pattern tree, we mine the top-k covering path rules. We maintain a buffer to store the top-k covering rules sorted according to their significance values. The significance value of the k-th covering rule is set as the significance threshold. This significance threshold is utilized to guide the subtree pruning in the path pattern tree during the generation of path rules. A subtree can be pruned if it will not generate a more significant rule than the k-th covering rule.

Four pruning strategies are introduced as follows.

1. **Support pruning.** If the support of a projected table is less than min_sup, the subtree under this projected table can be pruned. This is because the number of moving object ids in projected table decreases as the path pattern tree depth increases.

2. **Confidence pruning.** We can derive the significance upper bound of a projected table based on Lemma 11. If this significance upper bound is less significant than the current significance threshold, the subtree under this projected table can be pruned.

3. **Significance pruning.** If a projected table only has one nonempty column, the subtree under this projected table can be pruned. This is because the confidence

reaches the maximal value 1.0, but the support is non-increasing. This subtree will not generate a more significant path rule.

4. **Top-k covering rules pruning.** For each column C in a projected table, if the significance upper bound of column C is less than all significance thresholds of the trajectory ids in that column, this subtree can be pruned.

Algorithm 17 gives the process to mine the top-k covering path rule set. Lines 1-3 initialize a covering rules buffer and significance threshold pair (confidence, support) to be (0,0) for each trajectory. Line 4 creates an empty path pattern tree. Line 5 scans the vertices of trajectory network M to build the first level nodes of path pattern tree. The projected tables from root node to the first level nodes are created based on the class distribution of the first level nodes. Line 6 calls $DepthFirst()$ to build the path pattern tree in depth first search manner. Lines 7-8 obtain all covering path rules in buffer list and return them.

In procedure $DepthFirst()$, Line 11 checks the validity of projected table based on four pruning conditions. If any one condition holds, this subtree can be pruned. Otherwise, Lines 12-13 create a new node in tree and derive a candidate path rule from this node and its associated projected table. Line 14 checks the support of path rule. Lines 15-17 update the threshold as follows. If a trajectory T_n has m, where $m < k$, covering rules, the threshold remains unchanged; Otherwise, the threshold is equal to the significance of the k-th covering rule. Lines 18-20 create relevant edges under the current nodes, which are simply obtained from the trajectory network. Line 21 calls procedure $DepthFirst()$ to recursively search the longer path rules.

Algorithm 17: TopK Covering Rules Miner

```
input  : $D$: trajectory database;
         $M$: trajectory network;
         $min\_sup$: minimal support;
         $k$: number of covering rules for each trajectory.
output: $\mathcal{R}$: Top-$k$ covering path rule set.
1  foreach $T_n \in D$ do
2      Initialize a buffer $Buf\text{-}n$ and insert to buffer list $Buf$ ;
3      Initialize the pair (confidence, support) of $T_n$ to be (0,0);
4  Create a path pattern tree $PPTree$ with an empty root;
5  Scan $M.\mathcal{V}$ to build the first level nodes of $PPTree$;
6  Call DepthFirst($M$, $PPTree.root$, $min\_sup$, $Buf\text{-}k$);
7  $\mathcal{R}$=all path rules in $Buf$;
8  return $\mathcal{R}$
9  Procedure DepthFirst($M$, $nd$, $min\_sup$, $Buf$)
10 foreach projected table $PT_i$ of $nd$'s edge $e_i$ do
11     if $PT_i$ cannot be pruned by four pruning strategies then
12         Create a node $nd_i$ as the child node of $nd$;
13         Generate a path rule $\gamma_i$ from $nd_i$;
14         if $\gamma_i.sup \geq min\_sup$ then
15             foreach $T_n \in D$ covered by $\gamma_i$ do
16                 Insert $\gamma_i$ into in $Buf\text{-}n$;
17                 Update $T_n$'s significance threshold by $\gamma_i$;
18             foreach edge $e_{ij}$ of $nd_i$ in $M.\mathcal{E}$ do
19                 Create a tree edge of $nd_i$;
20                 Intersect $PT_i$ with the trajectory ids of $e_{ij}$ to obtain a projected
                   table $PT_{ij}$;
21             Call DepthFirst($M$, $nd_i$, $min\_sup$, $Buf$);
```

6.6 Duration-Aware Classifiers

In this section, we will present the strategies to build classifiers based on the duration-sensitive region rules and top-k path rules.

Given a path pattern $P = (mc_0 \xrightarrow{\alpha_1} mc_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_m} mc_m)$, its corresponding *centroid path* is $P_c = (c_0 \xrightarrow{t_1} c_1 \xrightarrow{t_2} \ldots \xrightarrow{t_m} c_m)$ where c_i is the centroid of mc_i and the t_i is the average duration of α_i.

The distance between rule γ and test trajectory T, denoted by $dist(\gamma, T)$, is defined as follows.

$$dist(\gamma, T) = \begin{cases} 0 & \text{if } \gamma \text{ is a region rule} \wedge T \in \gamma\ , \\ +\infty & \text{if } \gamma \text{ is a region rule} \wedge T \notin \gamma, \\ TD(\gamma.P_c, T) & \text{if } \gamma \text{ is a path rule.} \end{cases} \tag{6.21}$$

The classification score cScore indicates the degree of confidence that a path rule can be utilized to classify a trajectory. We consider two criteria in the cScore definition. First, the significant path rules (See Definition 20) will have the high cScore values. Second, the similar path rules in terms of trajectory distance will have the high cScore values. With these two criteria, we define the classification score of rule γ to trajectory T as follows:

$$cScore(\gamma, T) = \gamma.conf \times f(dist(\gamma, T)) \tag{6.22}$$

where $f(dist(\gamma, T))$ is a weighted function of trajectory distance $dist(\gamma, T)$, which can be any weighted function such as reciprocal function $f(x) = \frac{1}{x}$, quadratic function $f(x) = \frac{1}{x^2}$, Gaussian function $f(x) = \mathcal{N}(x; \mu, \sigma^2)$. In this work, we select Gaussian function.

Note that cScore does not include the support of path rules. This is because the path rules of high support and low confidence may have the large cScore values, which is inconsistent to the Definition 20.

We build two classifiers based on the valid region rules and top-k covering rule set as follows. The first classifier, named **Trajectory Classifier based on Features (TCF)**, is built by first transforming each trajectory into a cScore vector w.r.t. all the valid region rules and top-k path rules, and then feeding the vectors into a classification model.

The second classifier, named **Trajectory Classifier based on Region rules and Path rules (TCRP)**, is a rule-based classifier. For a test trajectory T, we calculate its cScore to all path rules, as given in Equation (6.22). We select the top-k path rules that have the highest cScore values and vote the class label of T to be the one with the majority vote. Note that the value k in TCRP is identical to the parameter k in top-k covering path rule set mining algorithm. If k=1, we only select one covering path rule in rule mining and predict the test trajectory based on the rule of the highest score. If k is equal to the total number of rules, we vote the test trajectory by all rules. The k value will be adjusted in the reasonable range according to the trajectory length. Intuitively, a longer trajectory implies more movement features, so a large k will be assigned for the long trajectories and a small k is assigned for the short trajectories.

6.7 Experimental Studies

In this section, we study the performance of TCF and TCRP. We use three real-life trajectory databases that were obtained from climate data, animals and vehicle objects, respectively. The details of each database are described as follows.

- **Hurricane track data**[1]. We use the Atlantic hurricanes between the year 1950 and 2008. The Saffir-Simpson scale classifies hurricanes into Scale 0 to 5, where a high scale indicates a high intensity. The numbers of trajectories (points) from Scale 0 to Scale 5 are 268 (5624), 150 (4434), 62 (2486), 73 (3199), 60 (2882), 26 (1208), respectively. We use the scales as class labels and isolate the Hurricane track data into three datasets. 1) Hurricane I contains trajectories of Scale 2 and 3; 2) Hurricane II contains trajectories of Scale 1 and 4; 3) Hurricane III contains trajectories of Scale 0, 4, 5, and we consider Scale 4 and 5 as one class label.

- **Animal movement data**[2]. The animal movement data has been generated by the Starkey project. We use the animal movements observed in June 1995. This data

[1]http://weather.unisys.com/hurricane/atlantic
[2]http://www.fs.fed.us/pnw/starkey/data/tables

set is divided into three classes by species: elk, deer, and cattle, whose numbers of trajectories (points) are 38 (7117), 30 (4333), and 34 (3540), respectively.

- **Vehicle track data** [3]. This dataset consists of the trajectories of 2 school buses and 50 trucks, which drive around Athens metropolitan area. The number of trajectories (points) are 145 (66096) and 276 (112203), respectively.

All algorithms are implemented in C++, and the experiments are carried on a server with dual Xeon 3GHZ processors and 4GB memory, running Window Server 2003 operating system. Table 6.1 summarizes the parameters used in TCRP classifier, where l is the length of the spatial plane.

Table 6.1: Summary of parameters

Symbols	**Range**	**Default**	**Descriptions**
min_sup	[0.01,0.02]	0.01	minimal support
min_conf	[0.8,0.9]	0.9	minimal confidence
k	1,3,5	3	top-k rules for voting
c	[0.01, 0.04]	0.02	initial cluster side/l
σ	[0.02, 0.1]	0.04	standard deviation of trajectory distance/l

6.7.1 Accuracy

We first evaluate the effects of region rules and path rules on the classification accuracy. We implement three versions of rule-based classifiers by incorporating three combinations of rules: TCR only incorporates region rules, TCP only incorporates path rules, and TCRP incorporates both region rules and path rules. Table 6.2 shows the classification accuracy of three rule-based classifiers on different datasets. We can see that TCR obtains the low accuracy on five datasets. This is expected because the region

[3]http://www.rtreeportal.org/

Table 6.2: Effects of rules on classification accuracy (%)

Datasets	TCR	TCP	TCRP
Hurricane I	48.14	60.00	60.00
Hurricane II	71.43	80.47	80.47
Hurricane III	68.07	75.42	79.66
Animal	73.52	81.37	82.35
Vehicle	81.00	95.80	96.56
Average	68.43	78.61	79.81

rules are not enough to distinguish the trajectories which have different moving directions. Based on Table 6.2, TCP obtains the higher accuracy than TCR on five datasets. This is because path rules are more discriminative than region rules on account of the significant moving features, including the speeds and directions, in path rules. Out of the three rule-based classifiers, TCRP obtains the highest accuracy due to an enhanced performance of both region rules and path rules on classification. The result of this experiment suggests to build rule-based classifiers by incorporating both region rules and path rules.

Next, we compare the classification accuracy of TCRP, TCF and the existing trajectory classifier RB-TB [42]. In principle, RB-TB and TCF transform trajectories into feature vectors and use existing classification methods to construct the classifier. In this experiment, we implement RB-TB and TCF based on SVM, C4.5 and BayesNet classification models. The parameter settings in RB-TB follow the reported values in [42]. Note that TCRP and TCF are built on both region rules and path rules, which are mined based on the default parameters: min_sup=0.01, min_conf=0.9 and k=5 for all datasets. All experiments are performed based on five-fold cross validation.

Table 6.3 summarizes the classification accuracy on five datasets. We can see that TCF is better than RB-TB on almost all datasets, and the TCF based on SVM obtains the highest average accuracy. This shows that incorporating duration information leads to more discriminative classifiers. Out of five datasets, TCRP obtains the highest accu-

Table 6.3: Effect of feature types on classification accuracy (%)

Datasets	SVM		C4.5		Bayes Net		Rule-based
	RB-TB	TCF	RB-TB	TCF	RB-TB	TCF	TCRP
Hurricane I	47.11	55.59	52.98	48.85	54.16	54.16	**60.00**
Hurricane II	75.23	77.14	72.38	73.81	74.76	77.14	**80.47**
Hurricane III	76.71	**81.72**	75.00	77.47	77.47	76.92	79.66
Animal	80.47	82.05	81.26	80.95	70.63	82.06	**82.35**
Vehicle	94.52	**98.44**	94.21	96.88	92.7	91.42	96.56
Average	74.81	78.99	75.17	75.59	73.94	76.34	**79.81**

racy on three datasets, Hurricane I and Hurricane II and Animal dataset. TCRP obtains the highest average accuracy. This shows that the rule based classifier is discriminative to handle trajectory classification. TCRP is also efficient because it is exempt from the conversion of the test trajectories into cScore vectors. In addition, TCRP is simpler than TCF and RB-TB because it does not need the involvement of the other classification methods. Due to these factors above, we consider TCRP to be an ideal trajectory classifier.

We show the discovered region rules and path rules by TCRP in Hurricane track data, Animal movement data and Vechicle track data as follows.

Rules in Hurricane Track Data

Figure 6.11, 6.12, 6.13 show the trajectory distribution, region rules and top-3 covering path rule sets for Hurricane I, II, III datasets, respectively. We can see that only few region rules are discovered on three datasets. This is because the distribution of two hurricane classes are quite similar for each dataset.

Rules in Animal Movement Data

Figure 6.14 shows the trajectory distribution of three animal species elk, deer and cattle in Animal database, and the valid region rules and the top-3 covering rules to build

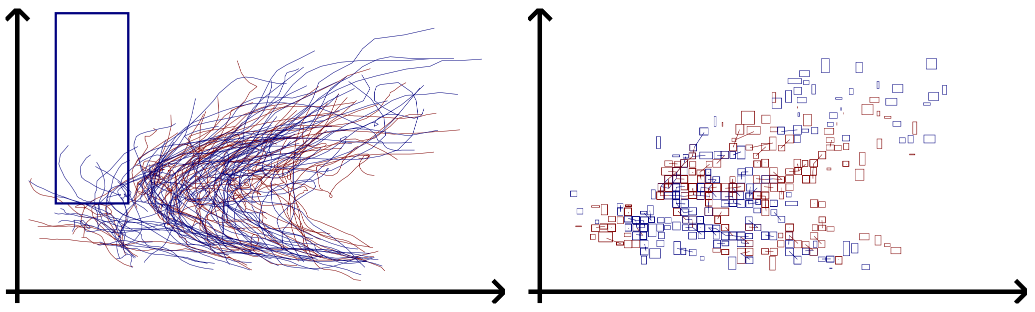

(a) Valid region rules (b) Top-3 covering path rules

Figure 6.11: Rules for Hurricane I dataset

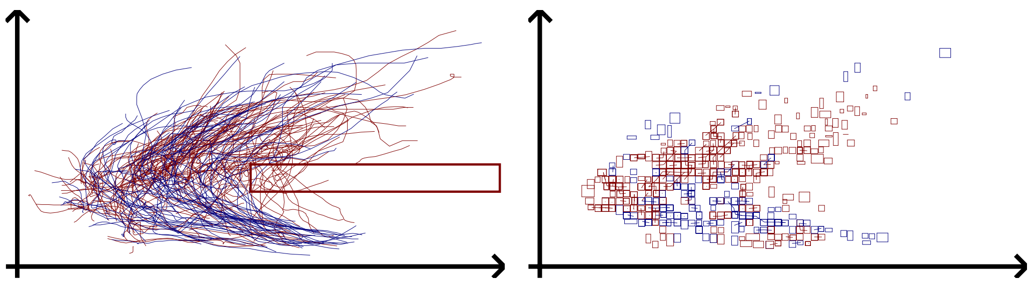

(a) Valid region rules (b) Top-3 covering path rules

Figure 6.12: Rules for Hurricane II dataset

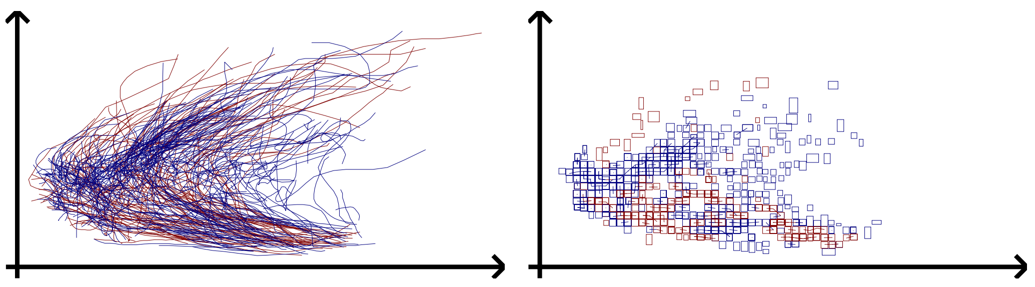
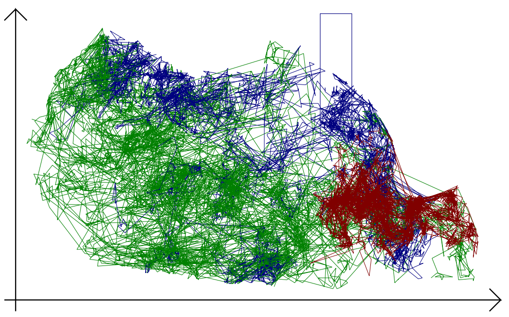

(a) Valid region rules (b) Top-3 covering path rules

Figure 6.13: Rules for Hurricane III dataset


(a) Valid region rules

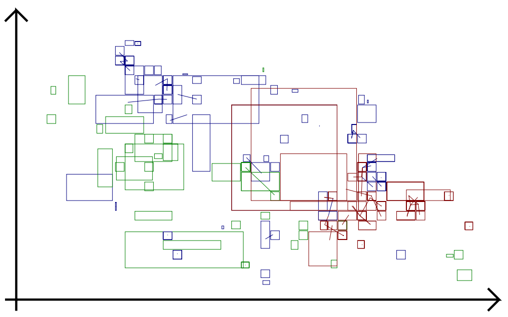
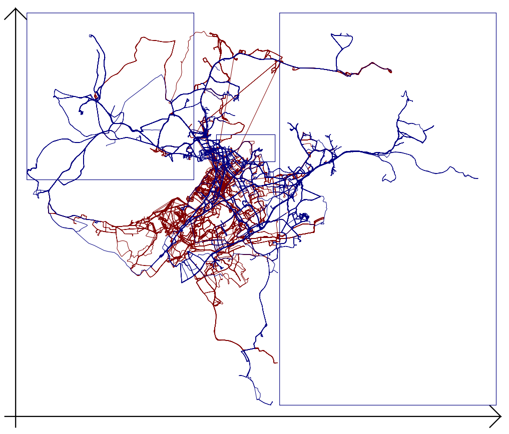

(b) top-3 path rules

Figure 6.14: Rules for Animal dataset

classifiers. We see that the trajectory distribution of three animal species are generally well separated. Thus, only one valid region rule is found. In the generated path rules, the granularity of micro-clusters is coarse and the path rules are rather short since they are already sufficient to discriminate among the different animal species.


(a) Valid region rules

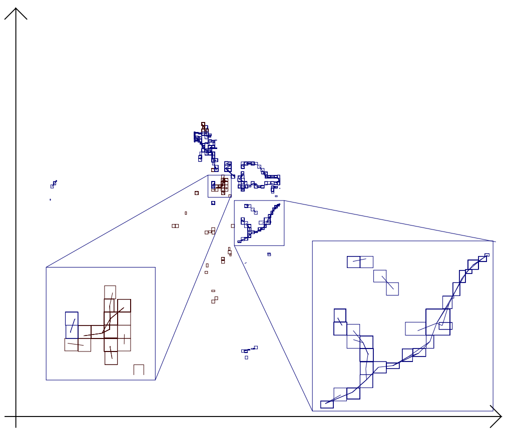

(b) Top-5 path rules

Figure 6.15: Rules for Vehicle dataset

Rules in Vehicle Truck Data

Figure 6.15 shows the trajectory distribution of buses and trucks on Athens city, and the three valid region rules and the top-5 covering path rules to build classifiers. We can see that two classes of trajectories are concentrated on the central region of the plane which is the urban area of Athens and all path rules occur in this area. The granularity of micro-clusters is rather fine and the path patterns contains multiple micro-clusters. This is because fine micro-clusters and long path patterns are more discriminative to handle similar data distributions of two classes in this area.

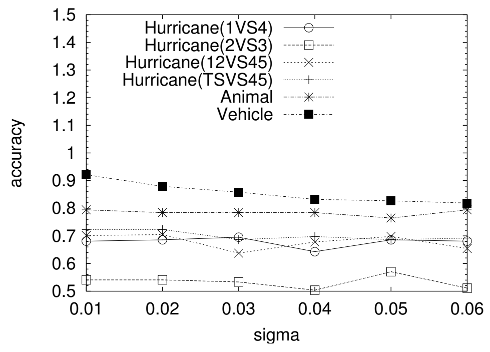
6.7.2 Sensitivity

We study the effect of σ on the accuracy of TCRP. σ controls the cluster size in the MDL procedure to mine the path rules. Small σ values result in the small clusters in trajectory network, thus the less number of path patterns and the shorter path patterns. Figure 6.16(a) shows the accuracy by varying σ. We can see that the accuracy curves reach maximal if σ varies in [0.02, 0.04], and the accuracy curves slowly decrease as σ increase after σ is greater than 0.03. This is because the standard deviations of sampling points in the five datasets range in [0.02, 0.04]. The trajectory networks which are trained and obtained in this range of σ properly model the trajectory database distribution.

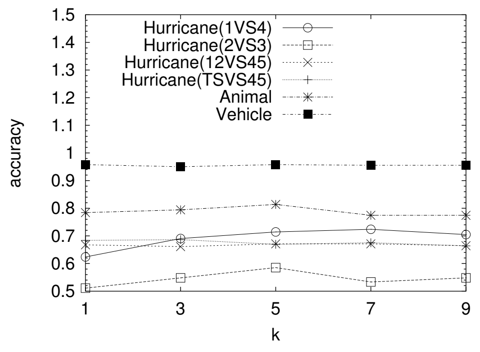
Next, we study the effect of k on the accuracy of TCRP. Figure 6.16(b) shows the accuracy by varying k. We observe that the accuracy curves reach the peaks when k is 3 or 5 for Hurricane datasets and Animal dataset. This result implies that a small k value is enough to distinguish the trajectories and a large k value may cause overfitting. For Vehicle dataset, the accuracy curves reach the peak when k is 5. This is because the trajectories in this dataset are longer than the others.

6.7.3 Efficiency

In this experiment, we evaluate the time efficiency of TCRP. We test the trajectory network training runtime by varying the initial grid side c, which has an important influence on the trajectory network training efficiency. In this experiment, σ is 0.04. Figure 6.17(a) shows the runtime of trajectory network training by varying c from 0 to



(a) Effect of σ on accuracy



(b) Effect of *k* on accuracy

Figure 6.16: Sensitivity

0.04, where *c*=0 means that the trajectory network training is performed based on the raw database. As we can see, the runtime decreases exponentially as *c* increases. This is expected because a larger *c* results in fewer number of sampling points, which decreases the trajectory network training time. Our experiments also show that the classification accuracy is stable if *c* varies in the range [0.01,0.03], which are less than σ=0.04. This suggests that setting *c* to be less than σ to train the trajectory network is efficient.

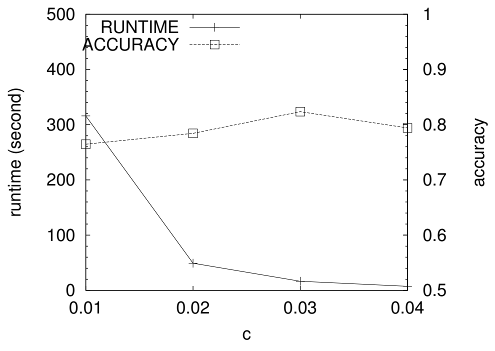
Next, we also test the efficiency of top-*k* covering rule miner by varying *min_sup* on five datasets. We fix *k*=5 and derive top-*k* covering rules on an existing trajectory network. Figure 6.17(b) and Figure 6.17(c) show the runtime on three Hurricane datasets, Animal dataset and Vehicle dataset. We observe that the runtime decreases as *min_sup* increases for all five datasets. This is expected because a large *min_sup* values trim the path pattern tree in early stages, so that a small number rules are selected as the valid rules or top-*k* covering rules.

Finally, we evaluate the effect of *k* in terms of efficiency. We fix *min_sup* to be 0.01 and run TCRP on all datasets. Figure 6.17(d) and Figure 6.17(e) show the runtime of mining top-*k* covering path rules on three Hurricane datasets, Animal dataset and Vehicle dataset. We can see that the runtime increases linearly as *k* increases. This is expected because a larger *k* results in a larger number of path rules generated.

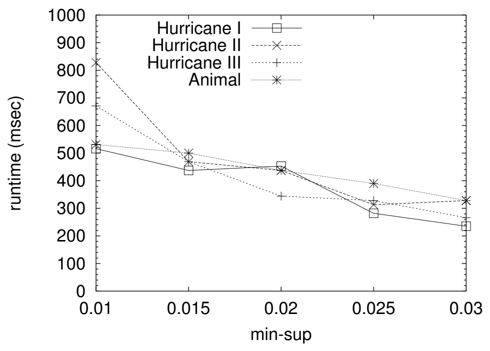
6.8 Summary

Trajectory classification is a very important problem in applications, and also a challenge research work in discovering and selecting the discriminative features for classification. Existing work [42] builds the classifier on shape-based features. In this work, we propose to discover the spatiotemporal features, including region rules and path rules, for classification. We utilize the influence model to present the trajectory distribution and design a space partition tree to facilitate the detection of valid region rules. To summarize the trajectory database, we introduce the concept of trajectory network, and we develop a trajectory clustering algorithm to compress the trajectory database into trajectory network of proper granularity. Based on the trajectory network, a path pattern tree is designed to enumerate all potential paths and facilitate the mining of top-*k* covering path rules. A few of pruning strategies are proposed to perform efficient path rules mining..

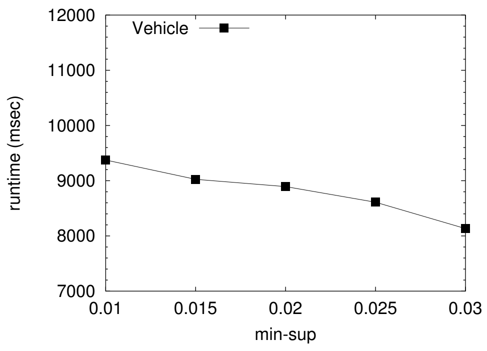
We build a hybrid classifier TCF which translates the trajectories into spatiotemporal feature vectors and feed in any classification model. We also build a rule-based classifier TCRP which predicts the class labels of trajectories by the region rules and path rules. All classifiers are tested on the real-life datasets. Experiments show that our classifiers obtain higher classification accuracy than the existing classifier in [42].



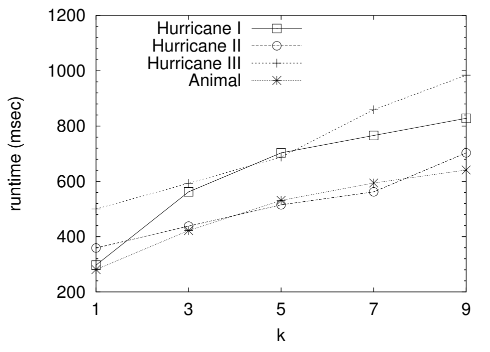
(a) Effect of *c* on Animal dataset



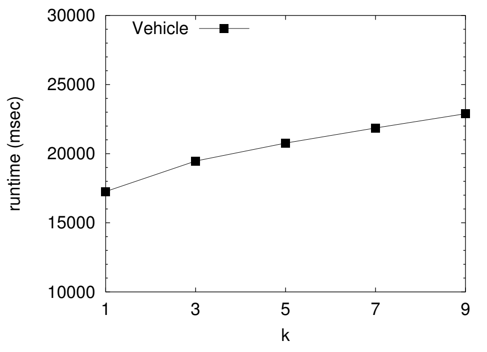
(b) Effect of *min_sup* on efficiency



(c) Effect of *min_sup* on efficiency



(d) Top-*k* covering path rules runtime



(e) Top-*k* covering path rules runtime

Figure 6.17: Efficiency

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we have investigated the spatiotemporal pattern mining in three types of spatiotemporal data. We have reviewed the current work in the area of sequential pattern mining, spatiotemporal data mining in event database and spatiotemporal data mining in moving object database. Although there has been a large amount of work in this area, there remains research challenges that need further investigation. This thesis has focused on three research problems.

The first research is to discover mutation chain in biological sequence data where each sequence is associated with location and time. We have proposed a mutation model where each sequence has influences to its nearby sequences. Based on the mutation model, we have introduced the notion of mutation chains to capture the subsequence changes over space and time. We have designed an integrated algorithm to mine mutation chains in a top-down search manner and have used two pruning strategies to reduce the search space. Experiments on synthetic datasets have shown that our algorithm is more scalable and more efficient than the base line algorithms. Experiments on real world Influenza A virus database have shown that our algorithms can be used to discover meaningful mutations.

The second research is to discover spatial interaction patterns in snapshot data. We have proposed an influence model for snapshot data where each object exerts influence to its nearby regions. We have defined the global Spatial Interaction Patterns (SIPs) on single snapshot, and have proposed a grid based influence model and have designed an algorithm called PROBER to discover SIPs based on a grid based influence model. Experiment results have demonstrated that the influence model based patterns effectively capture the spatial relationship of objects in snapshot data, and are easily extended to localized and time-associated patterns. We have extended SIPs to the Geographical-specific Interaction Patterns (GIPs) over continent snapshots, and have designed an algorithm called FlexiPROBER to discover the localized GIPs based on a quadtree based influence model. We also have developed an algorithm called MineGIC to discover three pattern trends, i.e., enlargement, shrinkage and movement of supporting regions, to capture the temporal changes in these patterns. Experiment results on both synthetic and real world datasets have shown that the proposed approaches are effective in mining the local geographical-specific interests patterns and discover their changes over time.

The last research problem is to discover duration-aware trajectory pattern in moving object data for trajectory classification. We have proposed to build trajectory classifiers that consider the duration of trajectories. We have introduced two kinds of features which incorporate duration information, duration-sensitive region rules and speed-differentiating path rules. The influences of moving objects to the regions are measured as the time spent by the moving objects in the regions. Based on this influence definition, we have utilized the top-down space partition method to mine the valid region rules. We have proposed the trajectory network to model the distribution of trajectories and employ MDL principle to evaluate the trajectory network. We have designed a path pattern tree to enumerate and mine the top-*k* covering path rules for classification. We also have built two classifiers TCF and TCRP to predict the class labels of test trajectories. Experiment results on real-world datasets have shown that both classifiers obtain higher classification accuracy than the existing classifier.

7.2 Future Work

There are a number of directions that require further investigation. We list three major directions for future work.

First, besides the physical geographical distances, the spatial constraints such as migratory bird patterns as well as modern air transportation routes, can be used to construct a spatial network to better model the spatial influence on the mutation likelihood. In addition, in road network based moving object databases, the object distances can be modelled by network distances instead of the geographical distances.

Another direction for future research is to investigate interesting spatial relationships such as spatial exclusion. Exclusion relationship refers to features that do not occur together, and no existing work focuses on spatial exclusion pattern mining. By enriching and mixing the spatial relationships, we will discover more useful and interesting knowledge in spatiotemporal data for real-world applications.

Finally, since spatiotemporal data comes from real application scenarios, they contain noise due to the limitation of measuring instruments and human recording errors. For example, the spatial positions of sampling points are imprecise, and the trajectories may miss some sampling points and insert some noise points. It is desirable to design a robust model which can handle the imprecise data and the trajectories of data inserting and deleting.

With more and more spatiotemporal data being tracked and analyzed in the real world, we believe this field will receive much attention in both academia and industry in the near future.

Bibliography

[1] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In Jorgeesh Bocca, Matthias Jarke, and Carlo Zaniolo, editors, *20th International Conference on Very Large Data Bases, September 12–15, 1994, Santiago, Chile proceedings*, pages 487–499, Los Altos, CA 94022, USA, 1994. Morgan Kaufmann Publishers.

[2] R. Agrawal and R. Srikant. Mining sequential patterns. In *ICDE*, page 3, Los Alamitos, CA, USA, 1995. IEEE Computer Society.

[3] A. Bairoch and R. Apweiler. The swiss-prot protein sequence data bank and its supplement trembl in 1999. *Nucleic Acids Research*, 27:49–54.

[4] Y. Bao, P. Bolotov, D. Dernovoy, B. Kiryutin, L. Zaslavsky, T. Tatusova, J. Ostell, and D. Lipman. The influenza virus resource at the national center for biotechnology information. *J. Virol.*, 82(2):596–601, 2008.

[5] Maurice Stevenson Bartlett. *The statistical analysis of spatial pattern*. Wiley, 1975.

[6] F. I. Bashir, A. A. Khokhar, and D. Schonfeld. Object trajectory-based activity classification and recognition using hidden markov models. *Image Processing, IEEE Transactions on*, 16(7):1912–1919, 2007.

[7] Huiping Cao, Nikos Mamoulis, and David W. Cheung. Mining frequent spatio-temporal sequential patterns. In *ICDM*, pages 82–89, 2005.

[8] Mete Celik, Shashi Shekhar, James P. Rogers, and James A. Shine. Mixed-drove spatiotemporal co-occurrence pattern mining. *IEEE Trans. Knowl. Data Eng.*, 20(10):1322–1335, 2008.

[9] Lei Chen and M. Tamer Ozsu. Robust and fast similarity search for moving object trajectories. In *SIGMOD*, pages 491–502, 2005.

[10] Hong Cheng, Xifeng Yan, and Jiawei Han. Incspan: incremental mining of sequential patterns in large database. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 527–532, New York, NY, USA, 2004. ACM Press.

[11] Gao Cong, Christian S. Jensen, and Dingming Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[12] Gao Cong, Kian-Lee Tan, Anthony K. H. Tung, and Xin Xu. Mining top-k covering rule groups for gene expression data. In *SIGMOD Conference*, pages 670–681, 2005.

[13] L. T. Daum, M. W. Shaw, A. I. Klimov, and L. C. Canas. Influenza a (h3n2) outbreak, nepal. *Emerg Infect Dis*, 11(8):1186–1191, August 2005.

[14] Philip M. Dixon. Ripley's k function. *Encyclopedia of Environmetrics*, 3:1796–1803, 2002.

[15] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2000.

[16] Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Cambridge University Press, 1999.

[17] R. C. Edgar. Muscle: multiple sequence alignment with high accuracy and high throughput. *Nucleic Acids Res.*, 32(5):1792–1797, 2004.

[18] Martin Erwig, Ralf Hartmut Güting, Markus Schneider, and Michalis Vazirgiannis. Spatio-temporal data types: An approach to modeling and querying moving objects in databases. *Geoinformatica*, 3(3):269–296, 1999.

[19] Wei Fan, Kun Zhang, Hong Cheng, Jing Gao, Xifeng Yan, Jiawei Han, Philip Yu, and Olivier Verscheure. Direct mining of discriminative and essential frequent patterns via model-based search tree. In *KDD '08: Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 230–238, New York, NY, USA, 2008. ACM.

[20] Ian De Felipe, Vagelis Hristidis, and Naphtali Rishe. Keyword search on spatial databases. In *ICDE*, pages 656–665, 2008.

[21] Elias Frentzos, Kostas Gratsias, and Yannis Theodoridis. Index-based most similar trajectory search. In *ICDE*, pages 816–825, 2007.

[22] Scott Gaffney and Padhraic Smyth. Trajectory clustering with mixtures of regression models. In *Knowledge Discovery and Data Mining*, pages 63–72, 1999.

[23] Minos N. Garofalakis, Rajeev Rastogi, and Kyuseok Shim. Spirit: Sequential pattern mining with regular expression constraints. pages 223–234. Morgan Kaufmann, 1999.

[24] Fosca Giannotti, Mirco Nanni, and Dino Pedreschi. Efficient mining of temporally annotated sequences. In *SDM*, 2006.

[25] Fosca Giannotti, Mirco Nanni, Fabio Pinelli, and Dino Pedreschi. Trajectory pattern mining. In *KDD '07: Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 330–339, New York, NY, USA, 2007. ACM Press.

[26] Karam Gouda and Mohammed Javeed Zaki. Efficiently mining maximal frequent itemsets. In *ICDM*, pages 163–170, 2001.

[27] Peter D. Gruwald, In Jae Myung, and Mark A. Pitt. *Advances in Minimum Description Length*. MIT Press, 2005.

[28] Sudipto Guha, Rajeev Rastogi, and Kyuseok Shim. Cure: an efficient clustering algorithm for large databases. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 73–84, New York, NY, USA, 1998. ACM.

[29] J. Han, J. Pei, and X. Yan. Sequential pattern mining by pattern-growth: Principles and extensions*. pages 183–220. 2005.

[30] Meng Hu, Jiong Yang, and Wei Su. Permu-pattern: discovery of mutable permutation patterns with proximity constraint. In *KDD '08*, pages 318–326, New York, NY, USA, 2008. ACM.

[31] Yan Huang, Shashi Shekhar, and Hui Xiong. Discovering colocation patterns from spatial data sets: a general approach. *IEEE Transactions on Knowledge and Data Engineering*, 16(12):1472– 1485, December 2004.

[32] Yan Huang, Liqin Zhang, and Pusheng Zhang. Finding sequential patterns from massive number of spatio-temporal events. In *SIAM Conference on Data Mining*, 2006.

[33] Yan Huang, Liqin Zhang, and Pusheng Zhang. A framework for mining sequential patterns from spatio-temporal event data sets. *IEEE Trans. on Knowl. and Data Eng.*, 20(4):433–448, 2008.

[34] Hoyoung Jeung, Qing Liu, Heng Tao Shen, and Xiaofang Zhou. A hybrid prediction model for moving objects. *Data Engineering, International Conference on*, 0:70–79, 2008.

[35] Hoyoung Jeung, Man Lung Yiu, Xiaofang Zhou, Christian S. Jensen, and Heng Tao Shen. Discovery of convoys in trajectory databases. *Proc. VLDB Endow.*, 1(1):1068–1080, 2008.

[36] Panos Kalnis, Nikos Mamoulis, and Spiridon Bakiras. On discovering moving clusters in spatio-temporal data. In *SSTD*, pages 364–381, 2005.

[37] AK. Kashyap, J. Steel, AF. Oner, and MA. Dillon. Combinatorial antibody libraries from survivors of the turkish h5n1 avian influenza outbreak reveal virus neutralization strategies. *Proc Natl Acad Sci U S A*, 105(598), 2008.

[38] Yiping Ke, James Cheng, and Wilfred Ng. Mining quantitative correlated patterns using an information-theoretic approach. In *KDD '06*, pages 227–236, New York, NY, USA, 2006. ACM Press.

[39] Eamonn J. Keogh. Exact indexing of dynamic time warping. In *VLDB*, pages 406–417, 2002.

[40] O.I. Kiselev, V. M. Blinov, and M. M. Pisareva. Molecular characteristic of influenza virus a h5n1 strains isolated from poultry in kurgan region in 2005. *Mol Biol (Mosk)*, 42(1):78–87, 2008 Jan-Feb.

[41] Krzysztof Koperski and Jiawei Han. Discovery of spatial association rules in geographic information databases. In M. J. Egenhofer and J. R. Herring, editors, *Proc. 4th Int. Symp. Advances in Spatial Databases, SSD*, volume 951, pages 47–66. Springer-Verlag, 6–9 1995.

[42] Jae-Gil Lee, Jiawei Han, Xiaolei Li, and Hector Gonzalez. Traclass: trajectory classification using hierarchical region-based and trajectory-based clustering. *Proc. VLDB Endow.*, 1(1):1081–1094, 2008.

[43] Jae-Gil Lee, Jiawei Han, and Kyu-Young Whang. Trajectory clustering: a partition-and-group framework. In *SIGMOD '07: Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 593–604, New York, NY, USA, 2007. ACM.

[44] Olive T Li, Michael C Chan, and Cynthia S Leung. Full factorial analysis of mammalian and avian influenza polymerase subunits suggests a role of an efficient polymerase for virus adaptation. *PLoS ONE*, 4(5):e5658, May 2009.

[45] Xiaolei Li, Jiawei Han, Sangkyum Kim, and Hector Gonzalez. Roam: Rule- and motif-based anomaly detection in massive moving object data sets. In *SDM*, 2007.

[46] Yifan Li, Jiawei Han, and Jiong Yang. Clustering moving objects. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 617–622, New York, NY, USA, 2004. ACM Press.

[47] Bing Liu, Wynne Hsu, and Yiming Ma. Integrating classification and association rule mining. In *KDD*, pages 80–86, 1998.

[48] Nikos Mamoulis, Huiping Cao, George Kollios, Marios Hadjieleftheriou, Yufei Tao, and David W. Cheung. Mining, indexing, and querying historical spatiotemporal data. In *KDD '04: Proceedings of the tenth ACM SIGKDD international*

conference on Knowledge discovery and data mining, pages 236–245, New York, NY, USA, 2004. ACM Press.

[49] A.C. Mishra, S.S. Cherian, and A. K. Chakrabarti. A unique influenza a (h5n1) virus causing a focal poultry outbreak in 2007 in manipur, india. *Virology Journal*, 6(26), 2009.

[50] Anna Monreale, Fabio Pinelli, Roberto Trasarti, and Fosca Giannotti. Wherenext: a location predictor on trajectory pattern mining. In *KDD '09: Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 637–646, New York, NY, USA, 2009. ACM.

[51] Yasuhiko Morimoto. Mining frequent neighboring class sets in spatial databases. In *KDD*, pages 353–358, 2001.

[52] Edward R. Omiecinski. Alternative interest measures for mining associations in databases. *IEEE Transactions on Knowledge and Data Engineering*, 15(1):57–69, 2003.

[53] J. Pei, J. Han, B. Mortazavi-Asl, and et.al. Prefixspan: Mining sequential patterns efficiently by prefix-projected patter growth. In *Proc. 2001 Int. Conf. Data Engineering*, pages 215–224, April 2001.

[54] Jian Pei, Jiawei Han, Behzad Mortazavi-Asl, Helen Pinto, Qiming Chen, Umeshwar Dayal, and Meichun Hsu. Prefixspan: Mining sequential patterns by prefix-projected growth. In *ICDE*, pages 215–224, 2001.

[55] Jian Pei, Jiawei Han, and Wei Wang. Mining sequential patterns with constraints in large databases. In *CIKM '02: Proceedings of the eleventh international conference on Information and knowledge management*, pages 18–25, New York, NY, USA, 2002. ACM.

[56] Jian Pei, Jiawei Han, and Wei Wang. Constraint-based sequential pattern mining: the pattern-growth methods. *J. Intell. Inf. Syst.*, 28(2):133–160, 2007.

[57] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[58] Brian D. Ripley. *Spatial Statistics*. Wiley, 1981.

[59] Jr. Roberto J. Bayardo. Efficiently mining long patterns from databases. In *SIGMOD '98: Proceedings of the 1998 ACM SIGMOD international conference on Management of data*, pages 85–93, New York, NY, USA, 1998. ACM Press.

[60] Dimitris Sacharidis, Kostas Patroumpas, Manolis Terrovitis, Verena Kantere, Michalis Potamias, Kyriakos Mouratidis, and Timos Sellis. On-line discovery

of hot motion paths. In *EDBT '08: Proceedings of the 11th international conference on Extending database technology*, pages 392–403, New York, NY, USA, 2008. ACM.

[61] Hanan Samet. *The design and analysis of spatial data structures*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.

[62] David W. Scott. *Multivariate Density Estimation: Theory, Practice, and Visualization*. Wiley, 1992.

[63] Shashi Shekhar and Yan Huang. Discovering spatial co-location patterns: A summary of results. pages 236–256. 2001.

[64] Chang Sheng, Wynne Hsu, and Mong Li Lee. Discovering geographical-specific interests from web click data. In *LOCWEB '08: Proceedings of the first international workshop on Location and the web*, pages 41–48, New York, NY, USA, 2008. ACM.

[65] Chang Sheng, Wynne Hsu, Mong Li Lee, and Anthony K. H. Tung. Discovering spatial interaction patterns. In *DASFAA*, March 2008.

[66] Chang Sheng, Wynne Hsu, and Mong li Lee. Discover spatiotemporal features for trajectory classification. In *Submitted to SIGKDD*, 2010.

[67] Chang Sheng, Wynne Hsu, Mong li Lee, Joo Chuan Tong, and See Kiong Ng. Mining mutation chains in biological sequences. In *ICDE*, 2010.

[68] A. C. C. Shih, T.-C. Hsiao, M. S. Ho, and W. H. Li. Simultaneous amino acid substitutions at antigenic sites drive influenza a hemagglutinin evolution. *Proc. Natl. Acad. Sci. USA.*, 104(15):6283–6288, 2007.

[69] Kyoko Shinya, Stefan Hamm, Masato Hatta, Hiroshi Ito, Toshihiro Ito, and Yoshihiro Kawaoka. Pb2 amino acid at position 627 affects replicative efficiency, but not cell tropism, of hong kong h5n1 influenza a viruses in mice. *Virology*, 320(2):258–266, 2004.

[70] Jeffrey S. Simonoff. *Smoothing Methods in Statistics*. Springer, 1996.

[71] Ramakrishnan Srikant and Rakesh Agrawal. Mining sequential patterns: Generalizations and performance improvements. In Peter M. G. Apers, Mokrane Bouzeghoub, and Georges Gardarin, editors, *Proc. 5th Int. Conf. Extending Database Technology, EDBT*, volume 1057, pages 3–17. Springer-Verlag, 25–29 1996.

[72] Ilias Tsoukatos and Dimitrios Gunopulos. Efficient mining of spatiotemporal patterns. In *SSTD*, pages 425–442, 2001.

[73] Petre Tzvetkov, Xifeng Yan, and Jiawei Han. Tsp: Mining top-k closed sequential patterns. *Knowl. Inf. Syst.*, 7(4):438–457, 2005.

[74] Florian Verhein and Sanjay Chawla. Mining spatio-temporal patterns in object mobility databases. *Data Min. Knowl. Discov.*, 16(1):5–38, 2008.

[75] Michail Vlachos, George Kollios, and Dimitrios Gunopulos. Discovering similar multidimensional trajectories. In *ICDE*, pages 673–684, 2002.

[76] Chuang Wang, Xing Xie, Lee Wang, Yansheng Lu, and Wei-Ying Ma. Detecting geographic locations from web resources. In *GIR '05: Proceedings of the 2005 workshop on Geographic information retrieval*, pages 17–24, New York, NY, USA, 2005. ACM.

[77] Junmei Wang, Wynne Hsu, and Mong Li Lee. A framework for mining topological patterns in spatio-temporal databases. In *CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 429–436, New York, NY, USA, 2005. ACM Press.

[78] Junmei Wang, Wynne Hsu, and Mong-Li Lee. Mining generalized spatio-temporal patterns. In *DASFAA*, pages 649–661, 2005.

[79] Junmei Wang, Wynne Hsu, Mong-Li Lee, and Jason Tsong-Li Wang. Flowminer: Finding flow patterns in spatio-temporal databases. In *ICTAI*, pages 14–21, 2004.

[80] Ke Wang, Yabo Xu, and Jeffrey Xu Yu. Scalable sequential pattern mining for biological sequences. In *CIKM*, pages 178–187, New York, NY, USA, 2004. ACM.

[81] Yida Wang, Ee-Peng Lim, and San-Yih Hwang. Efficient algorithms for mining maximal valid groups. *The VLDB Journal*, 17(3):515–535, 2008.

[82] Xifeng Yan, Jiawei Han, and Ramin Afshar. Clospan: Mining closed sequential patterns in large datasets. In *In SDM*, pages 166–177, 2003.

[83] Jiong Yang and Meng Hu. Trajpattern: Mining sequential patterns from imprecise trajectories of mobile objects. In *EDBT*, pages 664–681, 2006.

[84] Jiong Yang, Wei Wang, Philip S. Yu, and Jiawei Han. Mining long sequential patterns in a noisy environment. In *SIGMOD '02: Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, pages 406–417, New York, NY, USA, 2002. ACM.

[85] Jin Soung Yoo, Shashi Shekhar, and Mete Celik. A join-less approach for co-location pattern mining: A summary of results. In *ICDM*, pages 813–816, 2005.

[86] Jin Soung Yoo, Shashi Shekhar, Sangho Kim, and Mete Celik. Discovery of co-evoluting spatial co-located event sets. In *SDM*, 2006.

[87] Jin Soung Yoo, Shashi Shekhar, John Smith, and Julius P. Kumquat. A partial join approach for mining co-location patterns. In *GIS '04: Proceedings of the 12th annual ACM international workshop on Geographic information systems*, pages 241–249, New York, NY, USA, 2004. ACM Press.

[88] Mohammed J. Zaki. Spade: an efficient algorithm for mining frequent sequences. In *Machine Learning Journal, special issue on Unsupervised Learning*, pages 31–60, 2001.

[89] Tian Zhang, Raghu Ramakrishnan, and Miron Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data, Montreal, Quebec, Canada, June 4-6, 1996*, pages 103–114. ACM Press, 1996.

[90] Xin Zhang, Nikos Mamoulis, David W. Cheung, and Yutao Shou. Fast mining of spatial collocations. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 384–393, New York, NY, USA, 2004. ACM Press.

Appendix

Influence Approximation

Our idea to determine the appropriate resolution is as follows. First, we partition the plane into a coarse granularity. Then we recursively perform a split operation to divide each cell into 4 sub-cells. These sub-division steps will assign a finer granularity which is exactly half of the previous resolution. In this way, we can compute the effect of finer resolution, and eventually arrive at the appropriate resolution. Figure 7.1 shows the splitting strategy, where o is the position of object and p is the center of a big grid of side R, the distance from o to p is indicated by symbol d. After unform splitting, this big grid is partitioned into four subgrids, each of which has a side $R/2$. The distances from o to the center of each subgrid are d'_1, d'_2,d'_3,d'_4 respectively.

In the following analysis of the bounds of the approximation, we only consider the case where the objects are distributed in the east quarter area. Without loss of generality, any other distributions can be transformed into this case by rotating the cell.

After splitting, we will have the following equations according to the *Cosine Theorem*:

$$\begin{cases} d_1'^2 = d^2 + \frac{1}{8}R^2 - \frac{\sqrt{2}}{2}dR\cos\theta_1 \\ d_2'^2 = d^2 + \frac{1}{8}R^2 - \frac{\sqrt{2}}{2}dR\cos\theta_2 \\ d_3'^2 = d^2 + \frac{1}{8}R^2 + \frac{\sqrt{2}}{2}dR\cos\theta_3 \\ d_4'^2 = d^2 + \frac{1}{8}R^2 + \frac{\sqrt{2}}{2}dR\cos\theta_4 \end{cases}$$

Since $\theta_1 + \theta_2 = \pi/2$ and $\theta_3 + \theta_4 = \pi/2$, we have

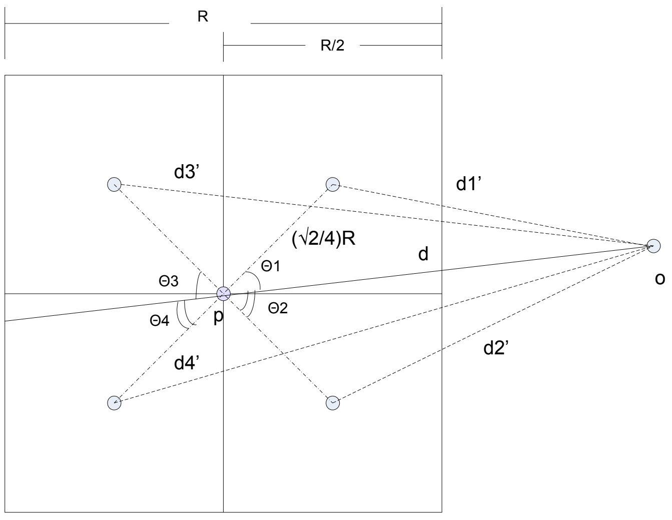


Figure 7.1: Cell splitting case

the following two constraints: $\begin{cases} 1 < \cos\theta_1 + \cos\theta_2 < \sqrt{2} \\ 1 < \cos\theta_3 + \cos\theta_4 < \sqrt{2} \end{cases}$, and $0 < \cos\theta_1, \cos\theta_2, \cos\theta_3, \cos\theta_4 < 1$.

The summation of the first two influence units is

$$\begin{aligned} & Inf_1 + Inf_2 \\ = & \frac{R^2}{4} \cdot (e^{-\frac{d_1'^2}{2\sigma^2}} + e^{-\frac{d_2'^2}{2\sigma^2}}) \\ = & \frac{R^2}{4} \cdot e^{-\frac{d^2}{2\sigma^2}} \cdot e^{-\frac{R^2}{16\sigma^2}} \cdot (e^{\frac{\sqrt{2}dR\cos\theta_1}{4\sigma^2}} + e^{\frac{\sqrt{2}dR\cos\theta_2}{4\sigma^2}}) \end{aligned} \tag{7.1}$$

Let $f(d) = e^{\frac{\sqrt{2}dR\cos\theta_1}{4\sigma^2}} + e^{\frac{\sqrt{2}dR\cos\theta_2}{4\sigma^2}} = e^{\frac{\sqrt{2}dR\cos\theta_1}{4\sigma^2}} + e^{\frac{\sqrt{2}dR\sin\theta_1}{4\sigma^2}}$, with $\theta_2 = \pi/2 - \theta_1$. $f(d)$ reaches a local minimal at $\theta_1 = 0$ and a local maximal at $\theta_1 = \pi/4$. So we have

$$2 < 1 + e^{\frac{\sqrt{2}dR}{4\sigma^2}} \leq f(d) \leq 2e^{\frac{dR}{4\sigma^2}}. \tag{7.2}$$

From Formula 7.1 and 7.2, we have

$$\frac{R^2}{2} \cdot e^{-\frac{d^2}{2\sigma^2}} \cdot e^{-\frac{R^2}{16\sigma^2}} < Inf_1 + Inf_2 \leq \frac{R^2}{2} \cdot e^{-\frac{d^2}{2\sigma^2}} \cdot e^{-\frac{R^2}{16\sigma^2}} \cdot e^{\frac{dR}{4\sigma^2}}$$

So the influence error at the first two sub-cells is

$$\begin{aligned} & IErr_{1,2}(\cdot) \\ = & \frac{\| (Inf_1 + Inf_2) - Inf/2 \|}{Inf_1 + Inf_2} \\ \leq & \frac{\frac{R^2}{2} \cdot e^{-\frac{d^2}{2\sigma^2}} \cdot (e^{-\frac{R^2}{16\sigma^2}} \cdot e^{\frac{dR}{4\sigma^2}} - 1)}{\frac{R^2}{2} \cdot e^{-\frac{d^2}{2\sigma^2}} \cdot e^{-\frac{R^2}{16\sigma^2}}} \\ = & e^{\frac{dR}{4\sigma^2}} - e^{\frac{R^2}{16\sigma^2}} \end{aligned} \tag{7.3}$$

Similarly, the summation of the last two influence units is

$$\frac{R^2}{2} \cdot e^{-\frac{d^2}{2\sigma^2}} \cdot e^{-\frac{R^2}{16\sigma^2}} \cdot e^{-\frac{dR}{4\sigma^2}} \leq Inf_3 + Inf_4 < \frac{R^2}{2} \cdot e^{-\frac{d^2}{2\sigma^2}} \cdot e^{-\frac{R^2}{16\sigma^2}}$$

So the influence error at the last two sub-cells are

$$IErr_{3,4}(\cdot) = \frac{\| Inf/2 - (Inf_3 + Inf_4) \|}{Inf/2} \leq 1 - e^{-\frac{R^2}{16\sigma^2}} e^{-\frac{dR}{4\sigma^2}} \tag{7.4}$$

Combining Formula 7.3 and 7.4, we have the influence error

$$IErr(\cdot) = \frac{IErr_{1,2} + IErr_{3,4}}{2} \leq \frac{1 - e^{-\frac{R^2}{16\sigma^2}} e^{-\frac{dR}{4\sigma^2}} + e^{\frac{dR}{4\sigma^2}} - e^{\frac{R^2}{16\sigma^2}}}{2} \tag{7.5}$$

As $0 \leq d \leq 3\sigma$, we normally substitute $k\sigma$ for d where $0 \leq k \leq 3$. So Formula 7.5 can be rewrote as

$$IErr(\cdot) \leq \frac{1 - e^{-\frac{R^2}{16\sigma^2}} e^{-\frac{kR}{4\sigma}} + e^{\frac{kR}{4\sigma}} - e^{\frac{R^2}{16\sigma^2}}}{2} \tag{7.6}$$

Merge vertices and edges

In this section, we give the computation processes of merging vertices and edges in TrajNet algorithm.

7.2.1 Merge Vertices

Figure 6.7 shows the process to merge vertices v_1, v_2 and v_3. Assume that all sampling points have the identical weight, the weight of three vertices are $W_1 = 4$ and $W_2 = W_3 = 2$, the radius of v_1, v_2 and v_3 are equal to 1.0, and σ=10.0. We have $H(v_1)$=0.81 and $H(v_2)$=$H(v_3)$=0.0, and c_v=0.0072. We consider the three cases as follows.

Case 1: Merge v_1 and v_2 to be a new vertex v_{12} which have the six sampling points and radius R_{12}=2. Its entropy $H(v_{12})$=1 and its weight W_{12}=6. In this case, the MDL_gain is $1 + H(v_1) + H(v_2) - H(v_{12}) + c_v(W_1R_1^2 + W_2R_2^2 - W_{12}R_{12}^2) = 1 + 0.81 + 0 - 1 + 0.0072 \times (4 \times 1^2 + 2 \times 1^2 - 6 \times 2^2) = 0.68$ bits.

Case 2: Merge v_1 and v_3 to be a new vertex v_{13} which have the six sampling points and radius R_{13}=2.5. Its entropy $H(v_{13})$=0.65 and its weight W_{13}=6. In this case, the MDL_gain is $1 + H(v_1) + H(v_3) - H(v_{13}) + c_v(W_1R_1^2 + W_3R_3^2 - W_{13}R_{13}^2) = 1 + 0.81 + 0 - 0.65 + 0.0072 \times (4 \times 1^2 + 2 \times 1^2 - 6 \times 2.5^2) = 0.93$ bits.

Case 3: Merge v_2 and v_3 to be a new vertex v_{23} which have the four sampling points and radius R_{23}=3. Its entropy $H(v_{23})$=1 and its weight W_{23}=4. In this case, the MDL_gain is $1 + H(v_2) + H(v_3) - H(v_{23}) + c_v(W_2R_2^2 + W_3R_3^2 - W_{23}R_{23}^2) = 1 + 0 + 0 - 1 + 0.0072 \times (2 \times 1^2 + 2 \times 1^2 - 4 \times 3^2) = -0.23$ bits.

Since Case 2 leads to a largest MDL_gain, we select to merge v_1 and v_3 to be a new vertex v_{13}.

7.2.2 Merge Edges

Figure 6.8 shows the process to merge edges. Assume that we have three edges e_1, e_2 and e_3, which move from vertex v_1 to vertex v_2. The Euclidean distance is $d(v_1, v_2)$=10.0. Assume that e_1 contains two red segments and their average duration is 2.0, e_2 contains one red segment and its duration is 1.0, e_3 contains one blue segment and its duration is 1.0. Let σ=10.0 and c_v=0.0072. We consider the three cases as follows.

Case 1: Merge e_1 and e_2 to be a new edge e_{12}. Its entropy $H(e_{12})$=0. Its weighted duration $t = \frac{2\times2.0+1\times1.0}{2+1} = 1.67$, thereby causing the distance error $d_1 = |1.67 - 2.0| \times 10.0 = 3.3$ and $d_2 = |1.67 - 1.0| \times 10.0 = 6.7$. In this case, the MDL_gain is $1 + H(e_1) + H(e_2) - H(e_{12}) + c_e(w_1 d_1^2 + w_2 d_2^2) = 1 + 0 + 0 - 0 + 0.0072 \times (2 \times 3.3^2 + 1 \times 6.7^2) = 0.52$ bits.

Case 2: Merge e_2 and e_3 to be a new edge e_{23}. Its entropy $H(e_{23})$=1. Its weighted duration $t = 1.0$, so $d_1 = 0.0$ and $d_2 = 0.0$. In this case, the MDL_gain is $1 + H(e_1) + H(e_2) - H(e_{23}) + c_e(w_2 d_2^2 + w_3 d_3^2) = 0$ bits.

Case 3: Merge e_1 and e_3 to be a new edge e_{13}. Its entropy $H(e_{13})$=0.92 and its weighted duration $t = \frac{2\times2.0+1\times1.0}{2+1} = 1.67$, thereby causing the distance error $d_1 = |1.67 - 2.0| \times 10.0 = 3.3$ and $d_2 = |1.67 - 1.0| \times 10.0 = 6.7$. In this case, the MDL_gain is $1 + H(e_1) + H(e_2) - H(e_{12}) + c_e(w_1 d_1^2 + w_2 d_2^2) = 1 + 0 + 0 - 1 + 0.0072 \times (2 \times 3.3^2 + 1 \times 6.7^2)$ = -0.48 bits.

Since Case 1 leads to a largest MDL_gain, we select to merge e_1 and e_2 to be a new edge e_{12}.