MINING PATTERNS IN COMPLEX DATA

DHAVALKUMAR PATEL

NATIONAL UNIVERSITY OF SINGAPORE

2011

MINING PATTERNS IN COMPLEX DATA

DHAVALKUMAR PATEL

(M.Tech.(Hons.),Indian Institute of Technology – Kharagpur, India)

A THESIS SUBMITTED FOR THE DEGREE OF

DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2011

Acknowledgments

I would like to express my sincerest gratitude to everybody who helped me throughout my time at NUS.

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. I have learnt a lot from them in many aspects of doing research. During the period of my graduate 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. Srinivasan Parthasarthy and Dr. Anthony Tung for providing research direction as a part of their class discussion. I thank Professor Limsoon Wong and Prof. Leong Tze Yun. As my thesis advisory committee members, they provided constructive advise on my thesis work. I also thank Professor Eammon Keogh for the fruitful discussions on time series data mining.

I would like to thank my family for their efforts to provide me the best possible educational environment. Last but not least, I would also like to thank my lab mates for providing a venue to discuss the idea.

Contents

Summary

Over the last decade, there has been an enormous growth in both the amount and the complexity of records that is collected and processed by humans and machines. This rapid growth has spurred interest in complex records that involve multiple kinds of data. Many applications from the clinical, surveillance and bioinformatics domains are now generating records with multiple kinds of data. For these applications to reach their full potential, we need to build effective techniques to analyze such complex records. Frequent pattern mining, a data mining technique, is widely used in data analysis and decision support. However, previous work has focused primarily on mining patterns from categorical data, numerical data, and sequence data. Very little attention has been paid to mine patterns from interval data, time series data and datasets with multiple kinds of data. In this work, we seek to develop techniques for analyzing complex records where each record is a combination of categorical, numerical, interval and time series data. Specifically, we address the following questions pertaining to mining patterns from complex records: How can we find frequent patterns from interval data? How can we discover frequent patterns from time series data? How can we mine frequent patterns from complex records having multiple kinds of data?

In the context of mining interval data, we investigate the problem of mining temporal patterns from interval-based event sequences. A temporal pattern is a sequence of events along with temporal relationships specified among events. First, we augment a well known hierarchical

representation with additional count information to model relationships among events in temporal pattern. This representation is lossless as the exact relationships among the events from temporal pattern can be fully recovered. Second, we propose an efficient algorithm to discover frequent temporal patterns from interval-based event sequences. Third, we demonstrate usability of discovered temporal patterns by building an interval-based classifier to differentiate closely related classes.

In the context of mining time series data, we examine the problem of discovering groups of motifs from different time series that exhibit some lag relationships. Time series motif is the recurring pattern in a single time series. First, we define a lag pattern that captures the invariant ordering among motifs where motifs are from different time series. Lag pattern characterizes localized associative pattern involving motifs derived from different time series and explicitly accounts for lag across multiple time series. Discovery of lag patterns requires to find motifs from each time series. We present an exact algorithm that integrates the order line concept and the subsequence matching property of the normalized time series to find all motifs of various lengths from single time series. Next, we propose a method to discover lag pattern efficiently. The proposed method utilizes inverted index and motif alignment technique to reduce the search space and improve the efficiency. Third, we show the usefulness of lag patterns discovered from a stock dataset by constructing stock portfolio that leads to a higher cumulative rate of return on investment.

In the context of mining dataset with multiple kinds of data, we introduce the notion of heterogenous pattern that captures the associations among patterns from different kinds of data. First, we present a unified algorithm that systematically discovers heterogenous patterns in a depth-first manner from a dataset consisting of categorical data, numerical, interval and time series data. Often times in many real-world problems frequent pattern mining algorithms yield many frequent patterns and only a subset of patterns are used in data analysis tasks such as classification. In view

of this, we present a sequential coverage based approach to discover an essential set of heterogenous patterns from dataset with multiple kinds of data. Experimental results on two real world datasets suggest that the proposed approach is efficient and can significantly improve the classification accuracy compared to existing classifiers.

List of Publications

This thesis is based on the following material:

- Dhaval Patel, Wynne Hsu and Lee Mong Li: Mining Relationships among Interval-based Events for Classification. *In Proc. of the 28th Special Interest Group on Management Of Data(SIGMOD)*, pages 98-108, 2008.

- Dhaval Patel, Wynne Hsu and Lee Mong Li: Finding Lag patterns from time series database. *In Proc. of the 22nd International Conference on Database and Expert Systems Applications(DEXA)*, pages 209-224, 2010.

- Dhaval Patel, Wynne Hsu and Lee Mong Li: Finding patterns from multiple kinds of data for classification. *Submitted to Special Interest Group on Knowledge Discovery and Data Mining(SIGKDD) for Review*, 2011.

Other publications based on material discussed in this thesis are:

- Dhaval Patel, Chidansh Bhatt, Wynne Hsu, Lee Mong Li and Mohan Kankanhalli: Analyzing Abnormal Events from Spatio-Temporal Trajectories. *In Proc. of the 8th International Workshop on Spatial and Spatiotemporal Data Mining(SSTDM)*, pages 120-128, 2009.

- Dhaval Patel: Mining Interval-orientation pattern from spatio-temporal database. *In Proc. of the 22nd International Conference on Database and Expert Systems Applications(DEXA)*, pages 190-209, 2010.

- Dhaval Patel, Wynne Hsu and Lee Mong Li: Discriminative Mutation Chain in Virus Sequence. *In Proc. of the 23rd International Conference on Tools with Artificial Intelligence(ICTAI)*, 2011.

- Sheng Chang, Dhaval Patel, Wynne Hsu and Lee Mong Li: Incorporating Duration Information for Trajectory Classification. *Submitted to International Conference on Data Engineering(ICDE) for Review*, 2012.

List of Figures

List of Tables

Chapter 1

Introduction

Recent advances in data collection and data storage technology have enabled business enterprises, research institutes and government agencies to accumulate enormous amounts of complex records from their daily activities. Fueled by an increasing need to rapidly analyze and summarize these data, researchers have turned to the field of knowledge discovery in databases. This field is concerned with development of efficient and effective techniques for data analysis and decision support. These techniques can yield valid, novel, potentially useful and understandable knowledge from the data. There are many kinds of knowledge that can be generated from data, e.g., frequent patterns, association rules, classification rules, clusters and outlier. A frequent pattern is a pattern that appears in a dataset with frequency no less than a user-specified minimum support threshold. Recent studies have shown that the frequent patterns are useful for mining associations [7, 62, 95], correlations [91], and many other interesting relationships among data [84, 51]. Moreover, it helps in data indexing [55], classification [51, 24], clustering [67], and other data mining tasks [41, 6] as well. Thus, frequent pattern mining has become an important data mining task.

1.1 Background

Frequent pattern mining was first proposed by Agrawal et al. in [8] to mine frequent itemsets from categorical data. For example, a set of items, such as milk and bread, denoted as {milk, bread}, that appears frequently together in categorical data, is a frequent itemset. Later, Agrawal and Srikant introduced a sequential pattern mining problem for sequence data [10]. This problem mines frequently occurring ordered events as patterns. A subsequence, such as buying first a PC, then a digital camera, and then a memory card, denoted as {PC $\rightarrow$ digital camera $\rightarrow$ memory card}, if it occurs frequently in sequence data, is a frequent sequential pattern. There have been hundreds of follow-up research publications, on various kinds of extensions and applications, ranging from scalable data mining methodologies [77, 19, 39, 42, 34, 44, 20], to handling a wide diversity of data types [46, 80, 13, 28, 55], various extended mining tasks [35, 63, 24, 6, 94, 67, 21], and a variety of new applications [51, 83, 22, 98, 29]. However, little attention has been paid on the study of mining frequent patterns from interval data, time series data and records that involve multiple kinds of data.

1.2 Motivation

While many of the frequent pattern mining algorithms are geared toward finding frequent patterns from categorical data, numerical data and sequence data, it has been noted recently that some of the database applications from the clinical, surveillance and bioinformatics domains have records with multiple kinds of data [43, 1, 4, 3, 2]. For example, a patient's record in hospital database typically comprises of categorical data, numerical data, interval data and time series data. One such example is the Stulong Dataset [1] obtained from Chiba Hospital. This dataset con-

tains complex records from the twenty years lasting longitudinal study of the risk factors of the atherosclerosis in the population of 1417 middle aged people. Table 1.1 presents a snapshot of this dataset. The overall objective in this dataset is to discover patterns that describe the risk factors for atherosclerosis cardiovascular disease. Similar examples of complex datasets include Hepatitis Dataset [1], Surveillance dataset [2], Drug Safety dataset [4] and etc. The challenge of analyzing these complex datasets is an immense task. Existing works can discover itemsets, numerical itemsets and sequential patterns from such complex datasets. However, mining patterns from interval data, time series data and complex data is also important.

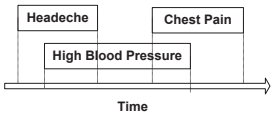
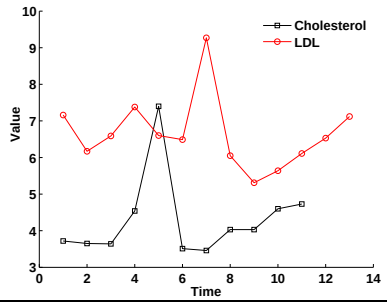
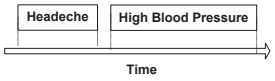
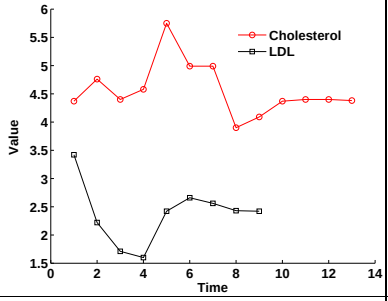
Id	Categorical Data	Numerical Data	Interval Data	Time Series Data	Class
1	Male, Smoking, Wine	Age = 21, DailyWineIntake = 2, AvgSysBldPre = 2			CVD = Yes
2	Male, NoSmoking, NoWine	Age = 30, DailyWineIntake = 0, AvgSysBldPre = 5			CVD = No
...	...	...	...	...	...

Table 1.1. Dataset with multiple kinds of data

1.2.1 Patterns in Interval Data

Interval data is a set of records, where each record is an ordered sequence of durative events, i.e. event with start time and end time. Existing pattern mining algorithms [10, 54, 14, 77] have focused on discovering frequent patterns from instantaneous events, that is, events with no duration. This assumption allows the discovered pattern to be simplified to an ordered sequence of events, such as $\{$Fever $\rightarrow$ Stomach ache $\rightarrow$ Vomit$\}$, $\{$Headache $\rightarrow$ High Blood Pressure$\}$, etc. However, such sequential patterns are inadequate to express the complex temporal relationships in domains such as medical, multimedia, meteorology and finance where the events' durations could play an important role.

For example, it has been observed that in many diabetic patients, the presence of hyperglycemia[1] overlaps with the absence of glycosuria[2], denoted as $\{$presence of hyperglycemia $\stackrel{Overlap}{\longrightarrow}$ absence of glycosuria$\}$. This insight has led to the development of effective diabetic testing kits [16]. In the case of dengue fever, knowing that there will be a decrease of platelet counts on the third day after the onset of fever, denoted as $\{$Onset of fever $\stackrel{Contain(3)}{\longrightarrow}$ decrease of platelet counts$\}$, has led to a better management of the disease. Clearly, there is a need for a mining algorithm that can discover complex relationships among events with duration, also known as interval-based events. As Allen's interval algebra [12] captures temporal relation between two events, we propose an lossless representation to encode Allen's temporal relation among more than two durative events. Furthermore, these discovered relationships can be used to build a classifier that is able to distinguish closely related classes and improve classification accuracy.

[1] high concentration of glucose in the blood
[2] presence of glucose in the urine

1.2.2 Patterns in Time Series Data

Time series data is a set of records, where each record is a sequence of regularly sampled real value observations. Time series analysis is an important research agenda at the heart of many applications drawn from diverse domains such as signal processing, statistics, medical, financial applications, etc. Analysis methods can be used to summarize or understand the underlying context or alternatively to make forecasts. Techniques can also be used to characterize associations across two or more time series. Many real world time series data exhibits associations among multiple time series. For example, a common association pattern observed in ICU monitoring is that an increase in PaO_2[3] level is followed by a decrease in $PaCO_2$[4] level during the patient's normal condition. Biologists are interested in identifying interesting regulator-regulated relationships in time-varying gene expression studies [40]. Health care practitioners are interested in observing the causal effect of daily air pollution level on mortality rate [52]. Doctors are interested in identifying important trigger patterns that highlight the relationship between heart rate and arterial blood pressure [33]. Furthermore, discovered associations might be helpful in a variety of applications, such as classifying human activities in smart home environment, implementing trading strategy in stock market, building classifier to differentiate closely related classes and so on.

Recent research interests in time series data mining mainly involve indexing time series for efficient similarity search [101], clustering time series [30, 38, 78], motif discovery [99, 57, 68, 88, 63, 27, 97], rule discovery [28, 62, 45] and time series correlation [102, 70], and so on. Similar to itemset in categorical data and sequential pattern in sequence data, **time series motif** is an important primitive in time series data. Time series motif is the recurring pattern(i.e., contiguous

[3]Partial Pressure of Oxygen
[4]Partial Pressure of Carbon Dioxide

subsequence) in single time series. For example, Figure 1.1 shows the time series of QLogic, Intel and JP Morgan stocks and highlights motifs m_1 = $\{\underline{s_{11}}, s_{12}, s_{13}\}$, m_2 = $\{\underline{s_{21}}, s_{22}, s_{23}\}$ and m_3 = $\{\underline{s_{31}}, s_{32}, s_{33}\}$ in the time series of QLogic, Intel and JP Morgan stocks respectively. Here, motif m_1 appears at time points 38, 55 and 112 in time series of QLogic stock.

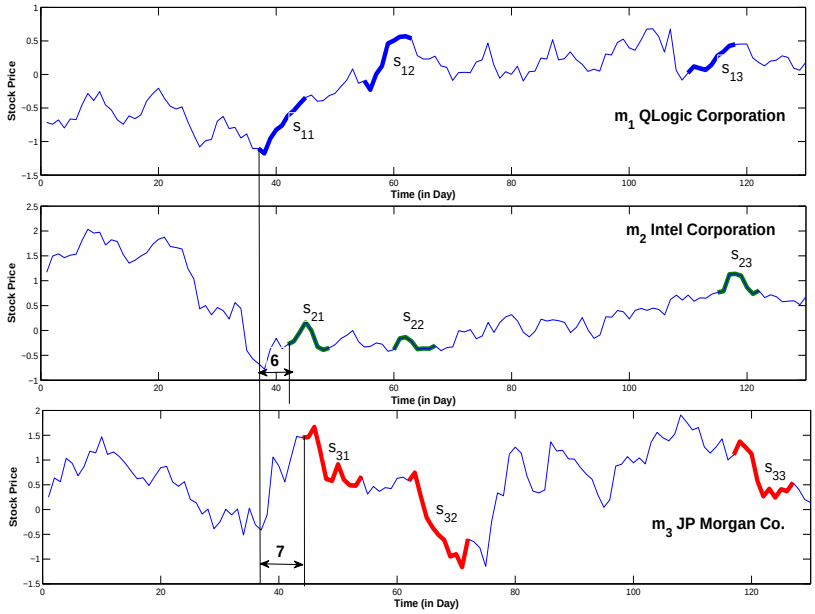
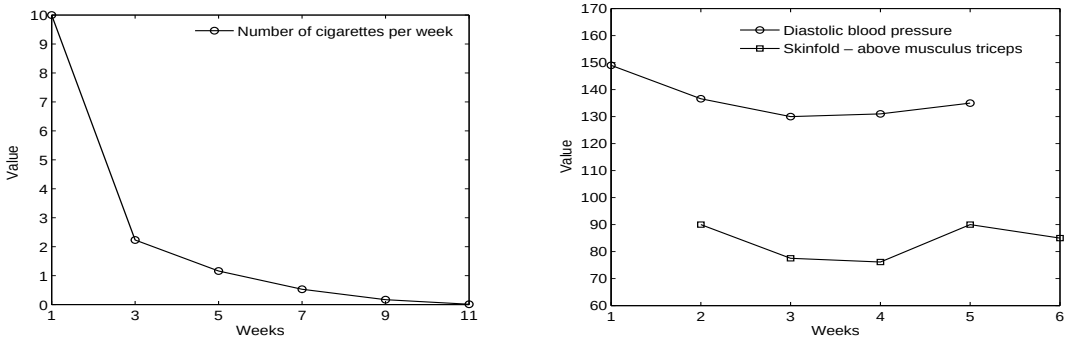


Figure 1.1. Lag relationships among motifs m_1, m_2 and m_3 reflecting competitor/co-operative behavior.

One useful association analysis of time series data involves finding repeated lag associations among motifs, where motifs are derived from different time series. For example, a closer examination of the motifs in Figure 1.1 reveals that the subsequences from one motif occur at a consistent lag relative to subsequences from other motifs. Here, s_{21} occurs with lag 6 relative to s_{11} while s_{31} occurs with lag 7 relative to s_{11}. This pattern is repeated for (s_{12}, s_{22}, s_{32}) and (s_{13}, s_{23}, s_{33}). In short, the lag relationship among the subsequences of motifs are repeated. The existence of such invariant ordering among the motifs suggests that there may exist some hidden

relationships. Further investigation[5] reveals that QLogic stock is competitor of Intel stock, while JP Morgan stock gives higher rating for investment in Intel Stock. Moreover, our experiments reveal that stock portfolio based on lag relationships among motifs leads to increase in the cumulative rate of return on investment.

Similarly, motifs and lag patterns from medical data can be used to characterize subgroup of patients having the same disease. For example, consider the motif given in Figure 1.2(a). This motif is present in 33 patients from the Stulong dataset. These patients reduced their intake of cigarettes from 10 cigarettes to 0 cigarette in two month period. Out of these 33 patients, only 8 patients develop cardiovascular disease. Clearly, smoking can be considered as a risk factor for cardiovascular disease. A pattern shown in Figure 1.2(b) captures lag relationship between Diastolic blood pressure and Skinfold. This pattern is observed in 12 patients having cardiovascular disease.



(a) time series motif (b) lag patterns

Figure 1.2. Example of motif and lag pattern obtained from Stulong Dataset

1.2.3 Patterns in Complex Data

Literature survey reveals that mining frequent patterns from categorical data [39], numerical data [86], and sequence data [77, 37, 18] separately has received lots of attention in recent years.

[5] Yahoo Finance - http://finance.yahoo.com

These pattern mining algorithms have also been extended to mine useful frequent patterns for classification [48, 98, 75, 24, 35] with improved classification accuracy. However, these algorithms only work for the specific kind of data they are designed for. Knowing the relationships(associations) among patterns from different kinds of data can aid in the understanding of a patient's health condition.

Consider the two patterns

$\{Male, Smoking\}$ and

$\{Headache \overset{Overlap}{\longrightarrow} HighBloodPressure\}$.

The pattern $\{Male, Smoking\}$ is a frequently occurring itemset [8]. Well-known algorithm such as FPTree [39] can be utilized to find such frequently occurring itemsets. On the other hand, the pattern $\{Headache \overset{Overlap}{\longrightarrow} HighBloodPressure\}$ is an interval-based temporal pattern and its discovery requires a totally different algorithm. Separately, these patterns may not raise any alarm as there are many male smokers in the population who go about their daily lives normally. Similarly, many people suffer from headache with elevated blood pressure but they do not experience any serious consequences. However, the combination of these two patterns reveal a different picture. Studies have shown that a male smoker who experiences headache with elevated blood pressure is a likely candidate for cardiovascular diseases. We call this combination of patterns from different kinds of data as **heterogenous** patterns. Our experiments on the real world datasets show that the heterogenous patterns discover previously unknown knowledge and significantly improve the classification accuracy.

1.3 Thesis Contributions

The complexity of data produced by applications is rapidly growing. Applications that produce and leverage complex datasets are becoming ubiquitous. Traditional frequent pattern mining and optimizations are essential for realizing efficient algorithms for analyzing complex datasets. The challenge of analyzing complex datasets is an immense task as much of the existing data mining work assume that the record consists of observations from a single kind of data. In this thesis, we investigate issues related to the analysis of datasets with multiple kinds of data. Such complex data is commonly found in applications in clinical, surveillance, bioinformatics and other domains. We first address the problem of mining frequent patterns from interval data and time series data. Later, we integrate frequent pattern mining algorithms of a single kind of data to mine frequent patterns involving multiple kinds of data. The contributions of this thesis are summarized below:

In the context of mining interval data, we investigate the problem of mining temporal patterns from interval-based event sequences. A temporal pattern is a sequence of events along with temporal relationships specified among interval events. First, we augment a well known hierarchical representation with additional information to model relationships among events in temporal pattern. This representation is lossless as the exact relationships among the events from temporal pattern can be fully recovered. Second, we propose an efficient algorithm to discover frequent temporal patterns from interval-based event sequences. The proposed algorithm has efficient candidate generation and support counting procedures and uses two optimizations to enhance the performance. Third, we demonstrate usability of discovered temporal patterns by building an interval-based classifier to differentiate closely related classes. This work is published in [75].

In the context of mining time series data, we examine the problem of discovering groups of motifs from different time series that exhibit some lag relationships. Time series motif is the recurring pattern in single time series. First, we define a lag pattern that captures the invariant ordering among motifs from different time series. Lag pattern characterizes localized associative pattern involving motifs derived from each time series and explicitly accounts for lag across multiple time series. Second, we present an exact algorithm that integrates the order line concept and the subsequence matching property of the normalized time series to find all motifs of various lengths from single time series. Third, we also propose a method to discover lag pattern efficiently. The proposed method utilizes inverted index and motif alignment technique to reduce the search space and improve the efficiency. At last, we show the usefulness of lag patterns discovered from a stock dataset by constructing stock portfolio that leads to a higher cumulative rate of return on investment. This work is published in [74].

In the context of mining dataset with multiple kinds of data, we introduce the notion of heterogenous pattern that captures the associations among patterns from different kinds of data. First, we present a unified algorithm that systematically discovers frequent heterogenous patterns in a depth-first manner from a dataset consisting of categorical data, numerical, interval and time series data. The proposed algorithm works in two stages. In the first stage, we mine frequent patterns from single kind of data. The second stage utilizes the output of the first stage to generate heterogenous patterns. In many real-world problems frequent pattern mining algorithms generate many frequent patterns and only a subset of patterns are used in data analysis tasks such as classification. Thus, we also present a mining strategy to discover the essential set of heterogenous patterns from dataset with multiple kinds of data. Our proposed algorithm is an iterative algorithm that discovers an essential set of heterogenous patterns for classification. In each iteration, we discover an essential

heterogenous pattern for classification and performs instance elimination. The instance elimination reduces the problem size progressively by removing training instances which are correctly covered by the discovered essential heterogenous pattern. Our experimental results suggest that our approach is efficient and can significantly improve the classification accuracy. This work is submitted to conference for review [73].

1.4 Organization

The rest of this thesis is organized as follows. Chapter 2 describes the fundamental concepts of frequent pattern and present an overview of the current research activities in this area. These activities can be divided based on data type such as categorical data, sequence data, interval based event sequence and time series.

Chapter 3 presents the formal definition of temporal pattern in interval data. We describe the problem of finding temporal patterns from interval based event sequence and proposed the algorithm in detail. The discovered temporal pattern is used in constructing the classifier.

Chapter 4 introduces a pattern template, called as lag patterns, to capture relationship among time series motif from different time series. We present an algorithm designed for mining the proposed patterns. We also construct a stock portfolio based on lag patterns and increase the cumulative rate of return on investment.

Chapter 5 introduces a pattern template, called as heterogenous pattern, to capture relationship among patterns from dataset with multiple kinds of data. We present two algorithms that mine complete frequent pattern and essential frequent patterns from dataset with multiple kinds of data. Finally, Chapter 6 presents concluding remarks about the thesis and provides suggestions for future research.

Chapter 2

Related Work

Data mining, a rapidly evolving area of research, aims to discover non-trivial, unknown and interesting knowledge from the large real data sets [23]. This discipline is an intersection of several disciplines such as statistics, databases, pattern precognition, optimization, visualization etc. Various techniques designed in this field such as association rule mining, pattern mining, clustering, classification etc have been successfully employed in various application domains such as e-commerce, biology, meteorology and many more. The data format varies from domain to domain and hence various pattern mining techniques are designed for different kinds of data. In this chapter, we review frequent pattern mining in categorical data, numerical data, sequence data, interval data, time series data and complex records.

2.1 Pattern Mining in Categorical Data

A pattern in categorical data is a collection of various items. For example, $\{$Bread , Butter$\}$ is a pattern. A pattern is interesting if it occurs frequently in data set. More specifically, a frequent pattern is a pattern with the support value higher or equal to the user defined minimum

support. A pattern {Bread , Butter} may be frequent pattern as it frequently bought by many customers together. Note that, frequent patterns are used to derive association rule, in other word, "Bread $\Rightarrow$ Butter" association rule is discovered from {Bread , Butter} frequent pattern. **Downward closure property** of frequent pattern is an important research contribution obtained in this area [10]. According to this property, if a pattern P is a frequent patterns then all of it's subsets must be the frequent. This property is utilized in various algorithms for pruning purpose. In particular, frequent pattern of length(size) k is used to derive frequent patterns of length(size) $k+1$. At this stage, items in frequent pattern do not have any ordering (i.e., {Bread , Butter} = {Butter , Bread }). Thus, pattern mining is also known as itemset mining. Existing algorithms can be classified into three categories: the Apriori algorithm [7, 9], vertical mining approach [85] and the pattern growth approach [19].

The Apriori algorithm uses candidate generate-and-test approach to explore the search space. Frequent itemset mining can be viewed as a set containment join between the transaction data and the search space of the frequent itemset mining problem. In the k-th pass of Apriori algorithm, the transaction database and the candidate k-itemsets are joined to generate frequent k-itemsets. The frequent k-itemsets are then used to generate the candidate $(k+1)$-itemsets.

The vertical mining approach maintains a tid list or a tid bitmap for each frequent itemset. Candidate itemset testing is performed by tid list/bitmap intersection. The main drawback of the vertical mining approach is that it needs to maintain a large number of tid lists/bitmaps, which prevents it from scaling well with respect to the number of transactions. Recently several techniques have been proposed to reduce the size of tid lists/bitmaps. Existing vertical mining algorithms usually change to horizontal mining if all the tid lists/bitmaps cannot be held in the memory.

The patten growth approach uses the depth-first order to probe the search space. It adopts the divide-and-conquer methodology. The search space is divided into disjoint sub search spaces. The database is partitioned according to the sub search space. For example, there are 5 items $\{a, b, c, d, e\}$ frequent in transaction database DB. The search space is divided into 5 disjoint sub search space: (1) itemset starting with a; (2) itemset starting with b, i.e. itemset containing b but not containing a; (3) itemset starting with c; (4) itemset starting with d; (5) itemset containing only e. Based on this five sub search space, the database is divided into 5 partitions. Each partition is called a conditional database. The conditional database of an itemset i, denoted as DB_i, contain all the transactions containing itemset i. Items before i are eliminated from each transaction in DB_i. Conditional database DB_i is used to discover superset of itemset i. This approach reduces the database scans.

A major challenge to mining frequent itemsets from a large data set is that such mining often generates a hugh number of itemsets satisfying the minimum support threshold, especially when threshold is low. To overcome this difficulty, the concept of closed itemset is introduced [100, 76]. An itemset X is closed if there exists no proper super-itemset Y such that Y has the same support count as X. Recent interest in data mining communities is to mine subset of essential itemsets that are useful for classification [24, 51].

2.2 Pattern Mining in Numerical Data

Quantitative association rules are introduced to mine patterns from numeric attributes [86, 6, 17, 90]. This approach dynamically discretized, i.e., binning, the numerical attribute during mining process so as to satisfy some mining criteria, such as maximizing the confidence or

compactness of the rules mined. Three common binning strategies can be utilized named equal width binning, equal frequency binning and clustering based binning. Very recently, an algorithm is proposed to mine subgroup discovery in numerical domains [41]. This approach uses sub-space clustering method to discover a subset of frequent numerical itemsets.

2.3 Pattern Mining in Sequence Data

Time plays an important role in understanding many real world phenomena and in sequence data it is used to order the underlying data. Pattern mining in sequence data is concerned with discovering the frequently recurring subsequences [10, 45, 36, 11, 62, 95]. For example , buying first a PC, then a digital camera, and then a memory card, denoted as $\{$PC $\rightarrow$ digital camera $\rightarrow$ memory card$\}$. Note that, items in pattern are ordered, so pattern is also known as **temporal pattern** or **sequential pattern**.

Various state of the art algorithms exist to discover frequent patterns from sequence data. The first few papers [10, 11] in this direction assume underlying sequence data has the form as shown in Table 2.1, which in our approach is referred as "sequence". Later, various algorithms [54, 31, 82, 15] were proposed by considering only single but long sequence as shown in Table 2.2 , which in our case is referred as "event sequence".

2.3.1 Set of Sequences

Agrawal and Srikant in [10] introduced a problem of mining frequently occurring subsequence pattern in the sequence database. A frequent sequential pattern is a sequence whose occurrence in the given data sets is higher than the user defined threshold (i.e., minimum support). In their

work, they have given a set of sequences where each sequence represents the customer transaction. A snapshot of such data set is given in the Table 2.1. In this example we have total three customers. Apriori algorithm is designed to handle this problem and also known as generalized sequential pattern (i.e., GSP) mining algorithm. GSP [87] is based on the breadth first principal and multi scan approach.

Customer	June 4	June 10	June 20
C1	Laptop	MemUpgrade	Audio
C2	DVD Rec	MemCard	USB Key
C3	-	Laptop	DVD-R

Table 2.1. Example : Sequence data

A number of extensions for sequential pattern mining has been proposed, for instance in [101] vertical list representation is used to speedup the time and in [42, 77] prefix based approach was proposed to speed up the mining algorithm for low minimum support value. One variant of the sequential pattern mining framework seeks to incorporate user's feedback in data mining process. For example, In GSP algorithm user can specify minimum and maximum time interval constraints between elements in a sequence. Recently, SPIRIT [36] (Sequential Pattern Mining with regular expression constraints) is proposed in order to mine sequential pattern based on the regular expression specified by the user. As PrefixSpan does not incorporate user's feedback in data mining task, GenPrefixSpan in [14] is proposed to overcome this limitation.

In [101], the authors proposed the SPADE algorithm. The main idea in this method is a clustering of the frequent sequences based on their common prefix and the enumeration of the candidate sequence. SPADE needs only three scans in order to extract the sequential patterns. The first scan aims at finding the frequent items, second scan aims at finding the sequence of length 2 and last one associates to frequent sequences of length 2, a Table of the corresponding sequence

id, itemset id in the data base. Based on this representation in the main memory, the support of sequence of length k is obtained by joining the Table of length k-1 prefix who can generate this pattern.

An interesting approach to mine sequential pattern aims at recursively projecting the data sequences into the smaller databases. Proposed by Han, FreeSpan in [42] is the first algorithm considering the pattern-projecting method for mining the sequential pattern. PrefixSpan [77] is the latest algorithm based on this concept. Starting from the frequent items of the databases, PrefixSpan generates projected database with the remaining data sequences. The projected database contains suffix of the data sequences from the original databases. The process is repeated on the projected databases.

Very recently, closed sequential patterns [89] and time lag pattern [37] are introduced. A sequential pattern P is closed if no supersequence of P has the same support as P. A time lag pattern is a sequence pattern with a time information between adjacent events in the sequence.

2.3.2 Event Sequence

A second class of approach, where Manila et al. in [54] introduce an episodic discovery framework from single long order sequence. In this framework, the author considers a long single sequence, such as alarms in telecommunication network and purpose is to discover the frequent temporal patterns. A snapshot of such data set is given in Table 2.2, where total five events $\{A, B, C, D, V\}$ and sequence length is 9. In their framework, long single sequence is known as event sequence and a temporal pattern as an episode. An episode can be a serial (i.e., A $\rightarrow$ B $\rightarrow$ C) or parallel (i.e., (ABC)). Apriori based mining algorithm is employed for this data (i.e., known as WINEPI). Two notions of frequency counting for an episode are discussed: fixed width overlap-

ping window (i.e., sliding window) and fixed width disjoint window. An automata-based counting scheme is also proposed to count the frequency of the episode. According to this scheme for serial episode, it takes O($l.k$) space and O($n.l.k$) time to compute the frequency of a set having total k elements, each element is of length l and n is the length of event sequence. They utilized the discovered frequent episode to analyze the alarm data and telecommunication data.

Time	1	3	4	5	7	11	12	14	15
Event	A	B	B	C	D	A	V	A	C

Table 2.2. Example : Event sequence data

A number of extensions for event sequence mining has been proposed and the detail can be obtained from the bibliography given in [81].

2.3.3 Set of Interval-based Event Sequences

The above two basic classes of problems have greatly fueled research in sequential pattern mining. Most of the data mining algorithms, such as Apriori, GSP, PrefixSpan, FreeSpan, SPADE consider events as instantaneous events(i.e., duration=0). Hence, all previous approaches mine sequential pattern considering only **"before"** and **"equal"** temporal relations between them. Recently, a few papers utilized duration of event directly or indirectly in sequential patter mining. Next we discussed various attempts made in this direction and problem with those approaches. A sample snapshot of one transaction we considered in our approach is given in Figure 2.1.

First, Kam et. al. [46] discovers frequent temporal patterns by considering the event duration. Independently, Panagiotis et. al. [72] proposed the H-DFS algorithm to mine frequent arrangement of temporal relation. Briefly, both approaches used vertical list concept similar to the SPADE algorithm with minor modification. In short, event sequence is transformed into a vertical represen-

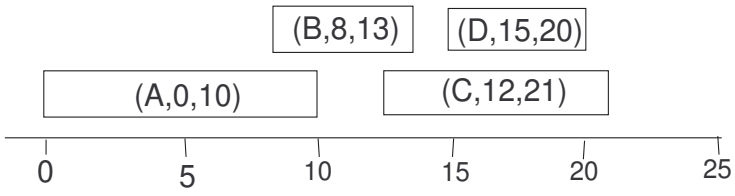


Figure 2.1. One record in interval data

tation (a.k.a., id-list), such as an event A's id-list contains $\{tid, [starttime, endtime]\}$. Then, id-list of one event is merged with id-list's of other events to generate temporal pattern between them. Thus, the temporal pattern between A and B has a id list as $\{\ tid, < [A.starttime, A.endtime], [B.starttime, B.endtime] > \}$. This is the basic step in both approaches. New patters are generated by combining the current frequent patterns's id-list with the id-list of single frequent pattern. The problems with the both approaches are

- As the length of temporal pattern increases, the second part of id-list representation will also increase. It is true that, the length of id-list of longer temporal pattern is less, but still traversing long second part of id-list is time consuming process.

- New frequent pattern is generated by combining id-list of frequent pattern at current level with the id-list of single frequent event. It should be noted that length of id-list of single frequent event is still long. Scanning this long list does not give good performance. It is also obvious now that the pattern generation in their approaches is different than the traditional SPDAE algorithm. SPADE combines two patterns at the same level to generate the frequent pattern for next level, while both the above algorithm combine a frequent pattern at current level with the frequent pattern at level 2 or 1. [1]

[1] In Paper [72], H-DFS it is not clear which level they utilized for pattern generation.

- Discovered temporal pattern by Kam et. al. is ambiguous as pointed out in [93].

Hoppner et al. [13] considers the duration of an event to discover temporal rule from the single event sequence. In their approach, if similar events overlap or meet, the events are merged into a larger segment. Author also proposes the extension to define the constraint patters such as, if event A follows event B within 10 second then in next 20 second event C follow events A. Cohen [71] proposes an algorithm to discover significant composite fluent from the underlying event sequence. They apply the composite relation approach to represent the longer frequent sequence. A similar approach is used in [56] for mining composite relation from the movement of the video data. The works [71] and [56, 49] are based on the lossy hierarchical representation. In short, all these algorithms are based on the ambiguous pattern and also lack the performance.

Recently, Shin et. al. in [93] proposed an algorithm called TPrefix identical to PrefixSpan algorithm for mining non-ambiguous temporal pattern from interval-based events. Note that, Lossless hierarchical representation in our case means representation is non-ambiguous. To obtain the non-ambiguous pattern, first the given event sequence is converted into sequence. For example, the sequences given in Figure 2.2 and 2.3 is converted to $A+ < B+ < A- < B-)$ and $(A+ < B+ = A- < B-)$. $A+$ denotes the A's start time and $A-$ denotes A's end time. For complex sequence such as given in Figure 2.4 is converted to $(A+ < B+ = C+ < A- < B- < C-)$. This converted sequence is known as the temporal sequence in their approach. Clearly this conversion double the length of event sequence as well increase the complexity of string matching. It also increases the complexity if similar events overlap.

Next, TPrefix discovers the single frequent events from the temporal sequences obtained from the preprocessing stage and then project the data base for each frequent single event. Again it scans the projected database for each frequent events in tern. Then it generates all the possible

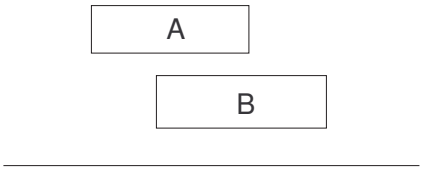


Figure 2.2. Example : Event sequence

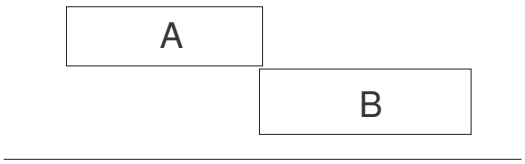


Figure 2.3. Example : Event sequence

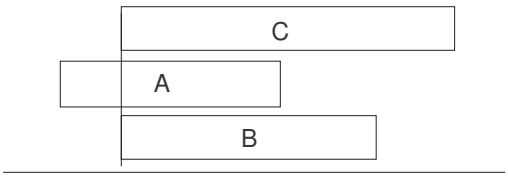


Figure 2.4. Example : Event sequence

candidates temporal relation (i.e., 13 allen's temporal relation) between temporal prefix of projected database and discovered frequent events. Then it counts the frequency of each generated candidates by scanning again the same projected data base. It means each projected database is scanned twice. First scan is comparatively easy compared to the second scan. The problems with this approach are

1. Candidate generation process does not have any candidate pruning strategy. This increase the support counting time. We need to generate the total 13 temporal relations from the one discovered frequent pattern.

2. Projected database are scanned twice. First scan is not complex. But the second scan required sophisticated string matching technique to speed the overall operations.

3. The intermediate representation of data occupied double memory compared to the original size data.

All the above problems have been validated with the original author of the paper. In short, these recent algorithms either require high main memory to store the prefix data set or lack effective candidate generation process or mine some spurious temporal patterns.

2.4 Pattern Mining in Time Series Data

Many real-world applications generate time series data, for example, intensive care unit monitoring, stock market, weather sensory instruments and many others. Various data mining methods have been proposed for clustering time series, finding lag based linear correlation between pairs of time series, discovering time series motifs and indexing time series for efficient similarity search. Very few work exist in the literature to discover patterns/rules from time series data. Based on underlying approach, they can be categorized into clustering of time series [28, 30, 38, 78], finding lag-based correlations in time series [102, 69], and motif discovery in time series [25, 50, 27, 57, 60, 59, 58, 99].

Das et. al. in [28] made an initial attempt to discover rules from time series by first obtaining all subsequences of length W from the given time series. These subsequences are then clustered using K-mean clustering algorithm. For each cluster, a representative pattern is obtained using the cluster center. With that, the original time series is transformed into a symbolic series(i.e., sequence) by using the cluster representative. Symbolic rules are then discovered from the symbolic series. The main limitation of this approach is that the rules discovered are all of length W.

Furthermore, the time complexity is too high for us to extend this approach to discover the desired lag associations across multiple time series.

Zhu et. al. in [102] aims to discover lag correlations among multiple streams in real time. They use fourier transform to summarize the streams and then compute their pairwise correlations. However, this method clearly misses any lag that is longer than the window length w. [84] discovers lag correlations among time series by using geometric probing. They calculate correlation values at various lags and use B-spline method to estimate the correlations for other possible lags. While this approach discovers lag correlations among the entire time series, it does not discover frequent lag correlations among subsequences.

Existing motif discovery approaches in time series are either approximate [25, 99, 57, 88] or exact [50, 64, 63]. In approximate motif discovery, time series is discretized into symbolic sequences and most recurring subsequences are discovered using variation of random projection based method [25]. Lin. et. al. in [50] introduce the notion of K-motifs, that is, a motif having K^{th} highest count of non-overlapping occurrences. The proposed algorithm hashes all subsequences into a table using their SAX word and then the promising buckets are processed to discover K-motifs. These works differ from ours in that they are approximate and dealing with fixed length motifs.

Recently, Mueen et. al. in [64, 63] propose algorithm to find the exact motifs efficiently by limiting the motifs to just pairs of time series that are very similar to each other. Both algorithms use order line and triangular inequality to reduce the distance computations. Their methods discover motifs of the given length. These works differ from ours in that their motif is a pair of the most similar subsequence.

There are also works that extend [25] to discover approximate multi-dimensional motifs from multiple time series [99, 57, 68, 88]. However, none of them considers time lag and invariant

ordering among motifs. Further, we do not adapt time series subsequences clustering method [28] to discover lag patterns, since clustering time series subsequence is meaningless as suggested in [32]. Our work aims to discover groups of motifs that exhibit some invariant ordering among the motifs within each group and explicitly capture the lag among them.

Minnen et. al. in [59] quantized time series globally and obtained motifs seeds of variable length by using a generalized suffix tree. However, global quantization tends to miss similar subsequences that have different amplitude. Oates in [68] developed the PERUSE algorithm to find recurring patterns in multivariate time series data. This algorithm works directly on real values and also extends the fixed length motifs to discover variable lengths motifs. This algorithm is not computational efficient as pointed out in [57].

In summary, none of the existing works is able to discover the repeated lag associations among motifs from different time series as motivated in our introduction.

2.5 Pattern Mining in Dataset with Multiple Kinds of Data

The works in [79, 80] discover multi-dimensional sequential patterns from categorical and sequence data. The algorithm discovers a pattern α from categorical (sequence) data and use α to discover patterns from sequence (categorical) data. Hence, α must contain patterns involving both categorical and sequence data. In our approach, we have relaxed this requirement. The works in [41, 65] mine mixture of categorical and numerical data based on subspace clustering. All these approaches aim to mine a complete set of patterns. Recent interest in data mining community is to discover a subset of patterns which are essentials for a particular data mining task [24, 35]. However, existing algorithms mine essential patterns only from records having single kind of data.

In summary, no methods in literature is designed to mine essential patterns from the records involving multiple kind of data for the classification task.

Chapter 3

Mining Patterns from Interval Data

In this chapter, we describe how to discover frequent temporal relationships that are hidden among events with duration. We have seen that the existing sequential pattern mining algorithms [10, 54, 14, 77] have focused on discovering frequent patterns from instantaneous events, that is, events with no duration. This assumption allows the discovered pattern to be simplified to an ordered sequence of events, such as "fever $\rightarrow$ stomach ache $\rightarrow$ vomit". However, such sequential patterns are inadequate to express the complex temporal relationships in domains such as medical, multimedia, meteorology and finance where the events' durations could play an important role. First, we present the following examples of how event duration is useful to find meaningful knowledge.

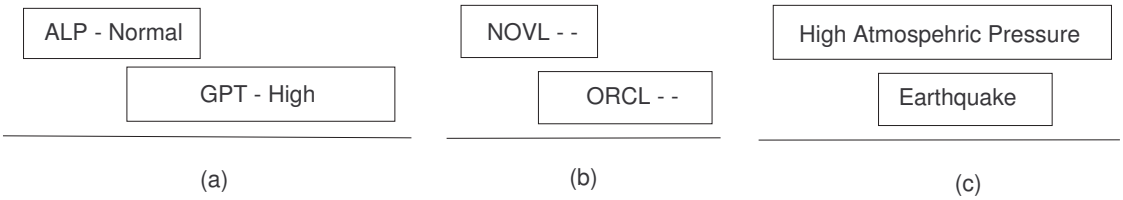


Figure 3.1. Example of temporal pattern (a) Medical domain (b) Financial data (c) Geological data

- **Medical Domain**: A physician has to analyze patient's data captured over a time in order to monitor a disease progress. For example, hepatitis patient data [1] contains information about 771 patients maintained over a period of 10 years. This dataset records the result of 230 in-hospital tests performed for each patient during each visit. For accurate data analysis and interpretation, temporal abstraction method is used to transform the raw values into interval based abstract description such as interval of "normal", "high" and "normalTohigh" for each test. Once interval-based abstraction data is obtained, physician looks for the existence of any predefined complex temporal pattern to interpret patient's behavior, such as temporal pattern shown in Figure 3.1(a). Here, ALP and GPT are two regularly conducted tests for hepatitis patient. Searching frequent temporal pattern in unsupervised manner helps physician to discover previously unknown complex temporal pattern. It is also possible to discover frequent temporal pattern which occurs only in patient with hepatitis B. Such pattern is useful for performing discriminative analysis.

- **Finance Domain**: In financial stock market, fluctuation of stock price can be modeled as interval-based events. For example, increase in stock A'price for at least three consecutive day is denoted as $A++$, decrease in stock A'price for at least three consecutive days is denoted as $A--$, and many more. Figure 3.1(b) shows an example of an interval based pattern for the stock market. The pattern reveals that, "Three days of consecutive increase of NOVL(Novel)'s price **overlap** with three days of consecutive increase of ORCL(Oracle)'s price". This pattern is useful to trader as they can easily incorporate it in trading strategy. We can also incorporate trader's preference in event modeling to perform user driven analysis.

Many other examples exist where use of event's duration results in more meaningful knowledge, such as "Earthquake event occurs only **during** High Atmospheric Pressure event" as shown in Figure 3.1(c). With the increasing amount of interval data, the urgency is to design an efficient mining algorithm that can discover frequent complex relationships among events with duration, also known as interval-based events. Furthermore, these discovered relationships could be used to build a classifier that is able to distinguish closely related classes.

In this chapter, we introduce the notion of temporal pattern in interval data and present an efficient pattern mining algorithm to mine frequent temporal patterns. We also build classifier using discovered temporal patterns to distinguish closely related classes. The rest of the chapter is organized as follows: Section 3.1 provides some preliminaries and definitions. Section 3.2 introduces the lossless representation of temporal pattern. Section 3.3 describes the IEMiner algorithm and the optimization strategies. Section 3.4 presents the design of IEClassifier. Section 3.5 gives the experiment results. We summarize in Section 3.6.

3.1 Preliminaries

An **event** is denoted by $E = (type, start, end)$, where $E.type$ denotes the type of event, $E.start$ and $E.end$ denote the event's start and end time respectively. For example, $(A, 1, 4)$ is an event of type A. It's start time is 1 and end time is 4.

An **ordered event list** $EL = \{E_1, E_2, \cdots, E_i, \cdots, E_n\}$ is a collection of events such that, all events in EL are sorted with respect to their start time. In case of conflict (i.e., two events start at a same time), an event which ends early is selected first. If still there is a conflict (i.e., events having same start and end times), we ordered them using their event's type. The length of EL, given

by $|EL|$, is the number of events in the list. For example, the length of first ordered event list in Table 3.1 is 4. In this chapter, we assume that events are always ordered in the event list.

Sr. No.	Ordered Event List
1	{(A,1,4), (B,2,5), (C,3,8), (D,6,7)}
2	{(A,1,2), (F,3,4), (G,5,6)}
3	{(A,1,4), (B,2,5), (C,3,8), (D,6,7), (F,9,10)}
4	{(A,1,3), (B,2,4), (D,5,6), (F,7,8), (G,9,10)}
5	{(Q,1,2), (C,3,4), (D,5,6)}
6	{(P,1,2), (C,3,4), (D,5,6)}

Table 3.1. Working Database DB

Relation	Interval Algebra	Dual Relation
E_i Before E_j	$(E_i.end < E_j.start)$	E_j After E_i
E_i Meet E_j	$(E_i.end = E_j.start)$	E_j Met-by E_i
E_i Overlap E_j	$(E_i.end > E_j.start) \wedge (E_i.end < E_j.end)$ $\wedge (E_i.start < E_j.start)$	E_j Overlapped-by E_i
E_i Start E_j	$(E_i.start = E_j.start) \wedge (E_i.end < E_j.end)$	E_j Started-by E_i
E_i Finished-by E_j	$(E_i.end = E_j.end) \wedge (E_i.start < E_j.start)$	E_j Finish E_i
E_i Contain E_j	$(E_i.start < E_j.start) \wedge (E_i.end > E_j.end)$	E_j During E_i
E_i Equal E_j	$(E_i.start = E_j.start) \wedge (E_i.end = E_j.end)$	E_j Equal E_i

Table 3.2. Temporal relationship between events E_i and E_j

Each event in an event list has a **temporal relationship** with all the other events in the event list. Table 3.2 shows the 13 temporal relationships defined by Allen [12] that can occur between any two interval-based events E_i and $E_j, i \neq j$.

Given two events E_i and E_j, a new **composite event** E is given as follow: E = ($(E_i.type$ R $E_j.type)$, $start$, $end)$, where R is temporal relationship between events E_i and E_j. The start and end times of E are given by minimum$\{E_i.start, E_j.start\}$ and maximum$\{E_i.end, E_j.end\}$

respectively. Here, $(E_i.type\ R\ E_j.type)$ is a representation of composite event E, denoted as $E.type$.

For example, E = (($A\ Overlap\ B$), 1, 5) is a composite event between events (A,1,4) and (B,2,5). Here, E's start time is 1 (i.e., minimum$\{$1,2$\}$) and an end time is 5 (i.e., maximum$\{$4,5$\}$). Also, ($A\ Overlap\ B$) is the representation of composite event E.

Given an ordered event list EL = $\{E_1, E_2, ... ,E_n\}$ of n events, a composite event of these n events is modeled using a hierarchical representation [46, 5, 66]. In this representation, first two events in EL are iteratively replaced by their composite event. This iterative process stops when only one event remains in EL.

For example, consider ordered event list EL = $\{$(A,1,4), (B,2,5), (C,3,8), (D,6,7)$\}$ of 4 events. To obtain composite event of EL, we first replace events (A,1,4) and (B,2,5) in EL by their composite event ((A Overlap B),1,4). Now, the updated EL is $\{$((A Overlap B),1,4), (C,3,8), (D,6,7)$\}$. Next, events ((A Overlap B), 1, 4) and (C,2,8) are replaced by their composite event. As a result, the updated EL is $\{$((A Overlap B) Overlap C), 1, 8), (D,6,7)$\}$. Finally, events ((A Overlap B) Overlap C), 1, 8) and (D,6,7) are replaced by their composite event. Now, resultant EL is $\{$((((A Overlap B) Overlap C) Contain D), 1, 8)$\}$. Here, ((((A Overlap B) Overlap C) Contain D), 1, 8) is composite event of EL.

Note that, hierarchical representation uses only subset of Allen's temporal relations while forming composite event. More specifically, we need only $\{$Overlap, Meet, Before, Contain, Finished-by, Equal, Start$\}$ temporal relations instead of original 13 temporal relations. While the traditional hierarchical representation provides an attractive and compact mechanism to express the temporal relations among events in composite event, the hierarchical representation of composite event is

lossless. Section 3.2 explains this problem in detail and presents the proposed augmented hierarchical representation.

Let database DB be a set of ordered event lists. A **temporal pattern** TP is of the form $E.type$ where $E.type$ is a representation of composite event. The support of temporal pattern TP, denoted as $sup(TP)$, is the number of event lists from DB that contain TP. The length of TP is given by the number of events in TP. In this chapter, we use n-pattern to denote a length n temporal pattern.

For example, "$((A\ Overlap\ B)\ Overlap\ C)$" is a temporal pattern. It is 3-pattern as it involves three events. Its support is 2, since only first and third event lists from dataset DB shown in Table 3.1 contain this pattern.

A **subpattern** of n-pattern is a representation of composite event consisting of k events from n-pattern, $k < n$. For example, consider 4-pattern P = $((A$ Overlap $B)$ Before $C)$ Before D. It's underlying event list is {A, B, C, D}. Here, $((A$ Overlap $B)$ Before $C)$ is one subpattern of P. Also, ($(A$ Overlap $B)$ Before D) is another subpattern of P. Note that, from n-pattern, we can generate n subpatterns of length $n - 1$.

A **temporal pattern** TP is **frequent** if $sup(TP) \geq minsup$. Given a minimum support threshold $minsup$ and interval data DB, we want to find the complete set of frequent temporal patterns. Note that, a temporal pattern satisfies the downward closure property, i.e., if temporal pattern TP is frequent, then all its subpatterns are also frequent.

3.2 Augmented Hierarchical Representation

Mining temporal patterns from interval data requires unique yet lossless representation to capture the temporal relationships among events in the temporal pattern. Hierarchical representation is widely used to encode the temporal relationships among more than two events[5, 66, 46]. However, this representation is lossy as explained below.

Assume, we have given a hierarchical representation "$((A$ Overlap $B)$ Overlap $C)$" of an ordered event list having three events. From this hierarchical representation, we can infer overlap relationship between events "A" and "B" as well as overlap relationship between events "B" and "C". But, temporal relationship between events "A" and "C" cannot be inferred. More specifically, temporal relationship between events "A" and "C" can be "before","meet" or "overlap" relation. Based on this discussion, Figure 3.2 represents three different interpretations of "(A overlap B) overlap C". We can observe that these three different structures have the same hierarchical representation. In other words, hierarchical representation "$((A$ Overlap $B)$ Overlap $C)$" is a lossy representation, as the encoded representation does not preserve the underlying temporal relationship among all the events. Any mining algorithm that is based on a lossy representation will lead to the discovery of many spurious patterns as non-frequent patterns may become frequent.

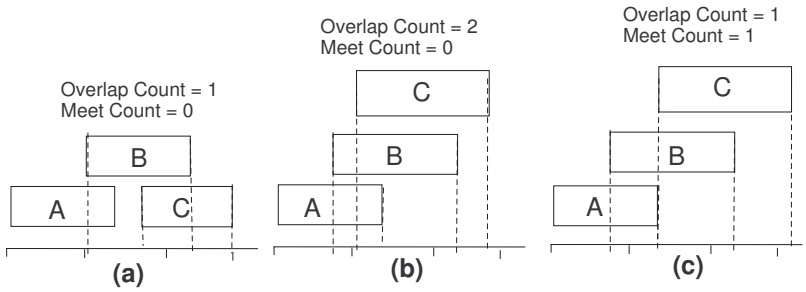


Figure 3.2. Same hierarchical representation "$(A$ Overlap $B)$ Overlap C" for three different event lists

To overcome this problem, we augment the hierarchical representation with additional count information. We observe that the first two cases in Figure 3.2 can be differentiated by using an *overlap count* to track the number of events in representation that actually have overlap temporal relationship with event C. For example, the *overlap count* for Figure 3.2(a) and 3.2(b) is 1 and 2 respectively. Figure 3.2(c) can be differentiated by using an additional *meet count* to indicate the number of events in representation that meets C.

An exhaustive enumeration shows that we need 5 variables, namely, *contain count c, finish_by count f, meet count m, overlap count o,* and *start count s* to differentiate all the possible cases. Figure 3.3 shows a partial listing of the various cases. We augment the hierarchical representation for a composite event E to include the count variable as follows:

$$\begin{aligned} E.type \quad = \quad & (((E_1.type\ R_1[c,f,m,o,s]\ E_2.type)\ R_2[c,f,m,o,s]\ E_3.type) \cdots \\ & \cdots\ R_{n-1}[c,f,m,o,s]\ E_n.type) \end{aligned}$$

We explain the proposed representation with examples. Consider Figure 3.2(a) as an event list. First, we obtain representation of composite event between events A and B, which results in ($A\ Overlap[0,0,0,1,0]\ B$). Next, we obtain representation between ($A\ Overlap[0,0,0,1,0]\ B$) and third event C. We observe that only one event (i.e., event B) in ($A\ Overlap[0,0,0,1,0]\ B$) actually overlaps an event C. Thus, overlap count = 1. Finally, representation of composite event between ($A\ Overlap[0,0,0,1,0]\ B$) and C is (($A\ Overlap[0,0,0,1,0]\ B$) $Overlap\ [0,0,0,1,0]\ C$). In Figure 3.2(c), we have one event from $(A\ Overlap[0,0,0,1,0]\ B)$ that has overlap temporal relationship with event C, and one event that meets event C. Thus, the final representation is (($A\ Overlap[0,0,0,1,0]\ B$) $Overlap[0,0,1,1,0]\ C$).

Thus, the representation of composite event in Figure 3.2 are represented as (see Figure 3.4)

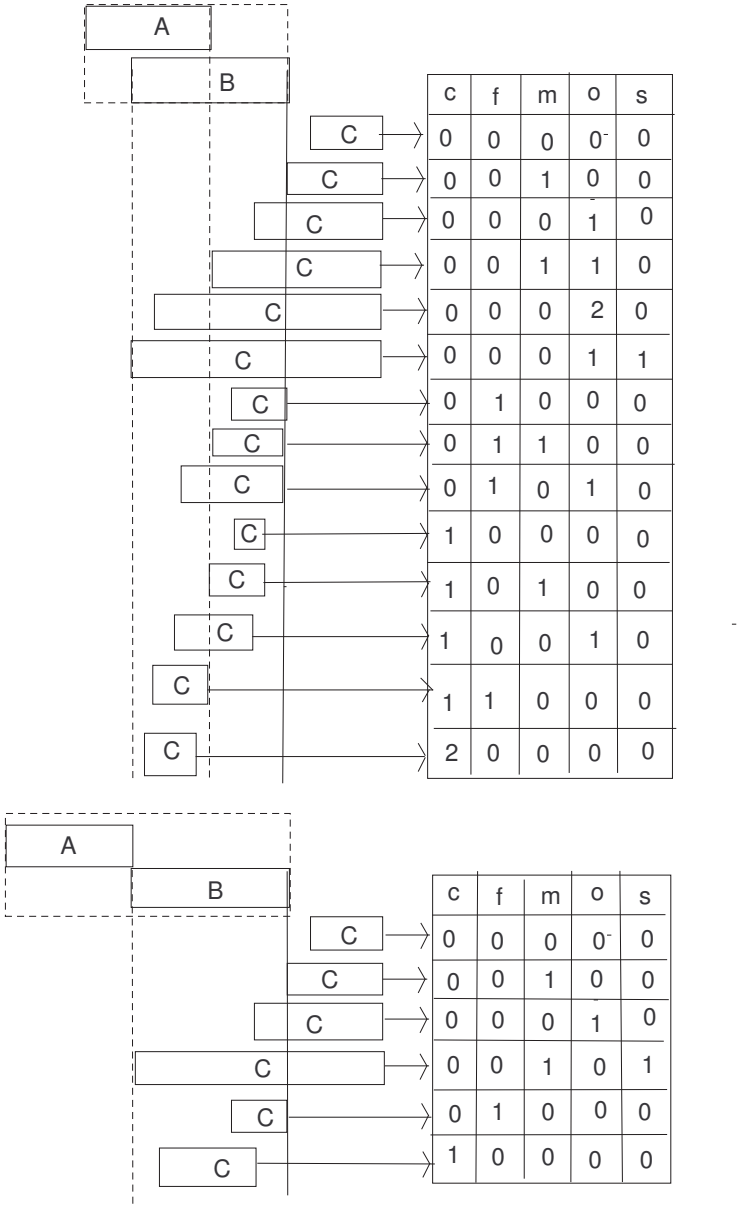


Figure 3.3. Partial enumeration of the possible cases involving 3 events

(A Overlap[0,0,0,1,0] B) Overlap[0,0,0,1,0] C

(A Overlap[0,0,0,1,0] B) Overlap[0,0,0,2,0] C

(A Overlap[0,0,0,1,0] B) Overlap[0,0,1,1,0] C

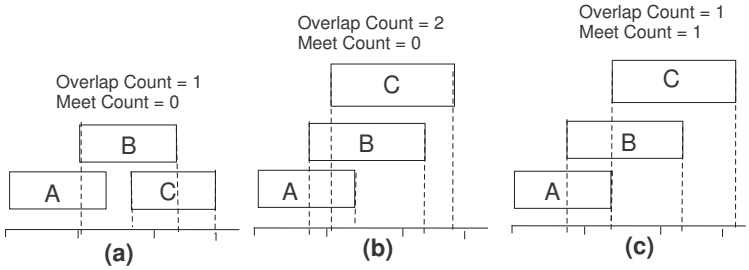


Figure 3.4. Example of augmented hierarchical representation: (a) (A Overlap[0,0,0,1,0] B) Overlap[0,0,0,1,0] C (b) (A Overlap[0,0,0,1,0] B) Overlap[0,0,0,2,0] C (c) (A Overlap[0,0,0,1,0] B) Overlap[0,0,1,1,0] C

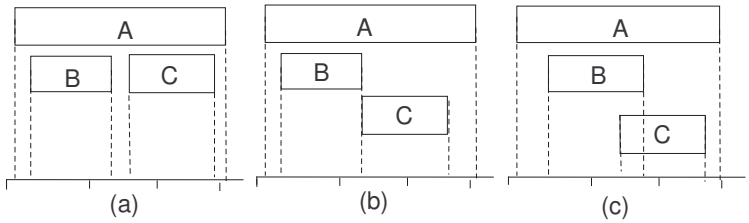


Figure 3.5. Example of augmented hierarchical representation

Similarly, hierarchical representation for Figures 3.5(a), 3.5(b) and 3.5(c) is represented as "(A Contain[1,0,0,0,0] B) Contain$[1,0,0,0,0]$ C", "(A Contain[1,0,0,0,0] B) Contain$[1,0,1,0,0]$ C" and "(A Contain[1,0,0,0,0] B) Contain$[1,0,0,1,0]$ C" respectively. Note that, the proposed representation does not require count information between first two events in the temporal pattern as well as when relation is $\{$Equal, Meet, Before$\}$ between temporal pattern and new event. More

examples to illustrate the augmented hierarchical representation are given in Figure 3.6 and Figure 3.7.

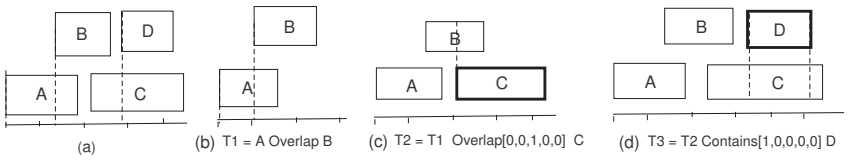


Figure 3.6. Examples of forming composite event with count variables.

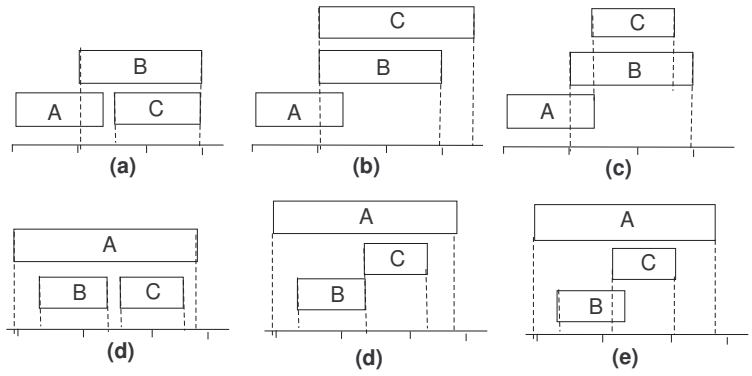


Figure 3.7. Example of temporal patterns (a) (A Overlap[0,0,0,1,0] B) Finished-by[0,1,0,0,0] C (b)(A Overlap[0,0,0,1,0] B) Overlap[0,0,0,1,1] C (c) (A Overlap[0,0,0,1,0] B) Contain[1,0,1,0,0] C (d) (A Contain[1,0,0,0,0] B) Contain[1,0,0,0,0] C (e) (A Contain[1,0,0,0,0] B) Contain[1,0,1,0,0] C (f) (A Contain[1,0,0,0,0] B) Contain[1,0,0,1,0] C

In order to prove that, the augmented representation is lossless, we use the concept of linear ordering of an event list. Given an ordered event list, a linear ordering is obtained by the chronological order of the start and end points of the events in the list. For example, the linear ordering of the first ordered list in Table 3.1 is :

$$\{\{A+\}\{B+\}\{C+\}\{A-\}\{B-\}\{D+\}\{D-\}\{C-\}\}$$

where $+$ indicates an event's start point and $-$ indicates an event's end point. A representation is lossless if we can recover the complete linear ordering of the start and end times of all the events which correspond to the underlying ordered event list.

Property I The augmented hierarchical representation is lossless.

Proof We prove the property using proof by induction.

Base case: A composite event consisting of two events is lossless. This inferred directly from the Interval algebra between two events given in Table 3.2.

Induction step: Suppose a composite event E_n consisting of n events is lossless. Let E_{n+1} = E_n $R[c, f, m, o, s]$ E be a composite event consisting of $n + 1$ events where E is a new event.

Since E_n is lossless, we can recover the linear ordering of the n events. With the new event E, we observe that the start time of E is constrained as follows:

1. $E.start$ = $E'.start$ for all E' where E' is an event in E_n and (E' Start E). The number of events that satisfy this condition is given by the *start count* s.

2. $E.start$ = $E'.end$ for all E' where E' is an event in E_n and (E' Meet E). The number of events that satisfy this condition is given by the *meet count* m.

3. $E.start < E'.end$ for all E' where E' is an event in E_n and (E' Overlap E) or (E' $Finished\_By$ E) or (E' Contain E) or (E' Start E). The number of events that satisfy this condition is given by the *overlap count* o, *finished_by count* f, *contain count* c and *start count* s.

Based on the above, we know that there are exactly c+f+o+s events whose end times come after the start time of E. Similarly, the end time of event E is constrained as follows:

1. $E.end < E'.end$ for all E' where E' is an event in E_n and (E' Contain E).

2. $E.end$ = $E'.end$ for all E' where E' is an event in E_n and (E' $Finished\_By$ E).

In other words, there are exactly c events whose end times come after the end time of E. With the linear ordering of E_n, we can determine the position of the start and end points of the new event E to obtain the linear ordering of E_{n+1}. Hence, we have shown that the augmented representation is lossless.□

3.3 Algorithm IEMiner

In this section, we present the proposed algorithm IEMiner (**I**nterval-based **E**vent **Miner**) to discover frequent temporal patterns from interval-based event sequences (see Algorithm 1). IEMiner follows an iterative approach known as level-wise search, where frequent temporal patterns of length $(k-1)$ are used to explore temporal patterns of length k. We first scan the database DB to obtain all the frequent events (Line 1). These frequent events are stored in $freSet_1$. In Lines 2-8, $freSet_{k-1}$ is used to find $freSet_k$. Here, $freSet_{k-1}$ denotes the set of frequent temporal patterns of length $k-1$, i.e., $(k-1)$-patterns. Obtaining $freSet_k$ from $freSet_{k-1}$ involves two basic steps:

- First, we call function $GetNextCandidateSet$ to obtain an initial $canSet_k$ from $freSet_{k-1}$ (Line 3). Here, $canSet_k$ denotes the set of k-patterns that can be frequent. We term this phase as **candidate generation**.
- Second, we identify the frequent temporal patterns from $canSet_k$. The $countSupport$ procedure is called for each event list EL in DB to determine the support count for each temporal pattern in $canSet_k$(Lines 4-6). Once all the EL in DB are processed, we obtain the frequent patterns and store them in $freSet_k$ (Line 7). We term this phase as **support counting**.

Algorithm IEMiner terminates when $freSet_{k-1}$ is empty, i.e., no frequent pattern is generated. Finally, all generated frequent temporal patterns are returned (Line 9).

Algorithm 1 Algorithm IEMiner

Input : Database DB; minimum support threshold $minsup$.
Output : frequent temporal pattern $freSet$

1: $freSet_1$ = {Scan database DB and obtain all frequent events}
2: **for** k=2; $freSet_{k-1} \neq \phi$; k++ **do**
3: $canSet_k \leftarrow$ **GetNextCandidateSet**$(freSet_{k-1})$
4: **for all** (event list $EL \in DB$) **do**
5: $canSet_k$ = **countSupport**$(k, EL, canSet_k)$
6: **end for**
7: $freSet_k \leftarrow \{c \in canSet_k \mid sup(c) \geq minsup\}$
8: **end for**
9: return $freSet = \cup_k freSet_k$

To address efficiency and scalability issues, IEMiner employs careful design of candidate generation(Line 3) and support counting procedures(Line 5). In addition to this, IEMiner also employs two additional optimistic strategies to further improve the performance. We have utilized database DB given in Table 3.1 to explain working of IEMiner algorithm. This dataset has total 6 event lists. Assume, $minsup$ is 2 event lists. Figure 3.8 illustrates the working of proposed algorithm for finding temporal patterns in DB. Now, we discuss candidate generation, support counting procedure and optimizations in detail.

3.3.1 Candidate Generation

In this section, we explain how $canSet_k$ is obtained from $freSet_{k-1}$. Algorithm GetNextCandidateSet(see Algorithm 2) gives the details of candidate generation process. Line 1 initializes the set of candidates to an empty set. We obtain the frequent 2-patterns from the set of frequent $(k-1)$-patterns in Line 2. Lines 3-10 generates candidate k-patterns from two frequent $(k-1)$-patterns and 2-pattern. All generated candidate patterns are returned(Line 11). The remaining section explains Line 6 in detail.

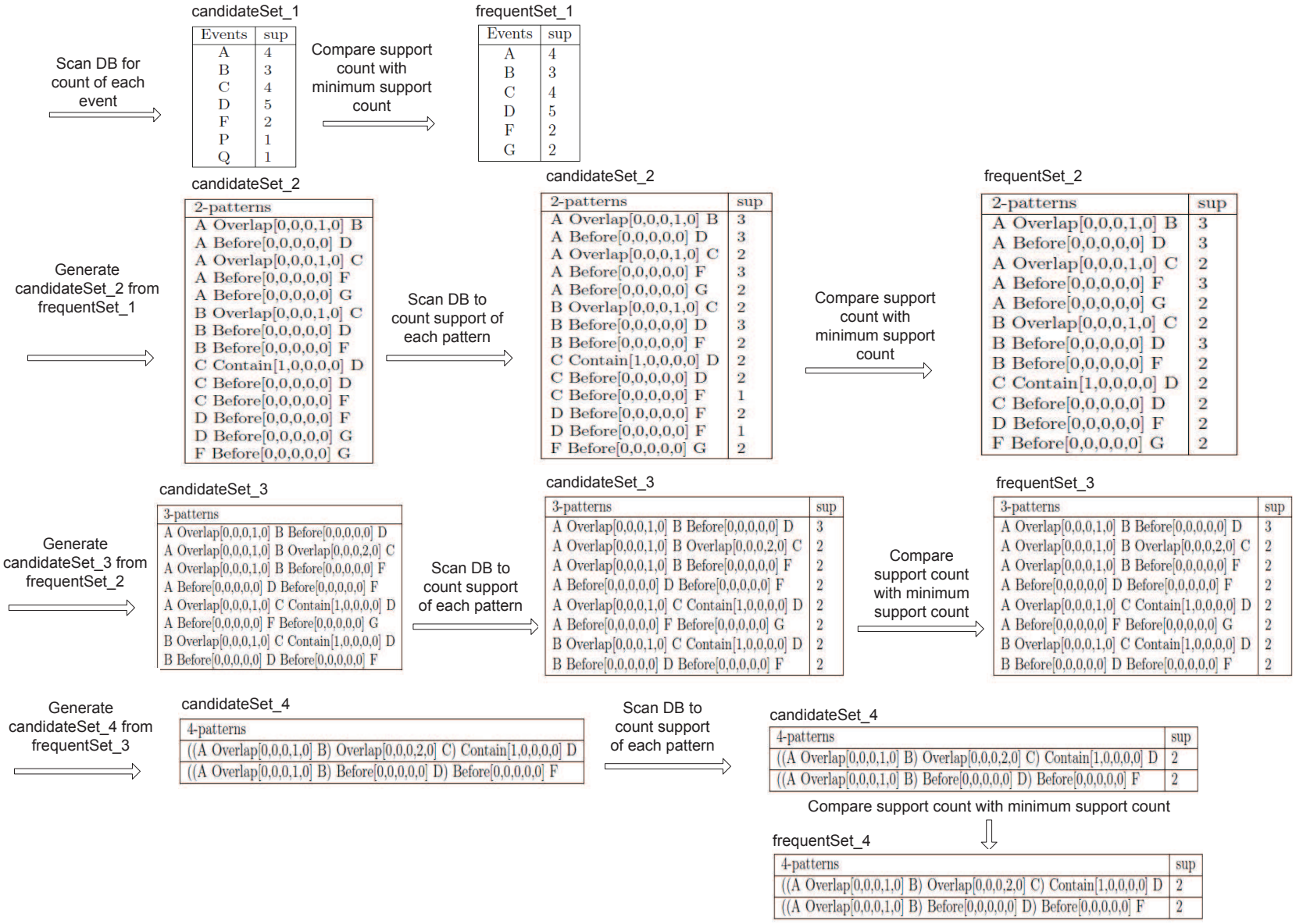


Figure 3.8. Generation of frequent temporal patterns, where the minimum support count is 2.

Algorithm 2 Algorithm GetNextCandidateSet

Input: $freqSet_{k-1}$, set of frequent $(k-1)$-patterns
Output: $canSet_k$, set of candidate k-patterns

1: $canSet_k \leftarrow \phi$
2: obtain useful 2-patterns $freSet_2$ from $freSet_{k-1}$
3: **for all** ($P \in f_{k-1}$) **do**
4: **for all** ($Q \in f_{k-1}$) **do**
5: **if** (prefix of P = prefix of Q) **then**
6: $tmp\_can \leftarrow$ {Generate k-patterns by **joining** P to Q using $freSet_2$}
7: $canSet_k \leftarrow canSet_k \bigcup tmp\_can$
8: **end if**
9: **end for**
10: **end for**
11: return $canSet_k$

Existing Apriori-based algorithms use two frequent $(k-1)$-patterns from $freSet_{k-1}$ to generate single candidate k-pattern. More specifically, candidate k-pattern R is generated from frequent $(k-1)$-pattern P and frequent $(k-1)$-pattern Q if prefix of P = prefix of Q[1]. Next, according to downward closure property, if pattern R is frequent then all of it's subpatterns are also frequent. Based on this, pattern R will be pruned, if any of it's subpattern is not frequent. Note that, existing Apriori-based algorithms generate only single k-pattern from two $(k-1)$-patterns[2]. However, in our method, we need to generate multiple candidate k-patterns from two frequent $(k-1)$-patterns.

For example, consider joining frequent 3-pattern P = "(A Overlap[0,0,0,1,0] B) Overlap[0,0,0,2,0] C" to frequent 3-pattern Q = "(A Overlap[0,0,0,1,0] B) Before[0,0,0,0,0] D" as shown in Figure 3.9. Note that, prefix of P = prefix of Q[3]. From these two patterns, we can generate total "five" candidate patterns as shown in Figure 3.10. Generating multiple candidate patterns incurs

[1]A prefix of a pattern is the pattern with the end cut off. For example itemset {A,B} is prefix of itemset {A,B,C}. Similarly, {A $\rightarrow$ B} is a prefix of {A $\rightarrow$ B $\rightarrow$ C}

[2]Given frequent 2-itemsets {A,B} and {A,C}, we can generate candidate pattern {A,B,C}. Given frequent sequential pattern of length 2 {A $\rightarrow$ B} and {B $\rightarrow$ C}, we can generate candidate pattern {A $\rightarrow$ B $\rightarrow$ C}

[3]A prefix of n-pattern is a representation of composite event consisting of first $n-1$ events from n-pattern. For example, "(A Overlap[0,0,0,1,0] B)" is prefix of "(A Overlap[0,0,0,1,0] B) Overlap[0,0,0,2,0] C".

high cost on checking downward closure property. However, careful observation reveals that we do not need to generate first three patterns in Figure 3.10. Consider the first pattern in Figure 3.10. The temporal relationship between events C and D in this pattern is $C\ Meet[0,0,1,0,0]\ D$. Now, from $freSet_2$, a set of frequent temporal patterns of length 2, shown in Figure 3.8, we can verify that $C\ Meet[0,0,1,0,0]\ D$ is not frequent. Indirectly, first pattern cannot be frequent. Similar apply to second and third patterns. This example shows that, frequent temporal relationship between event C and D from $freSet_2$ can be used to prune the candidate patterns while joining P to Q. Note that, events C and D have latest start time in patterns P and Q respectively.

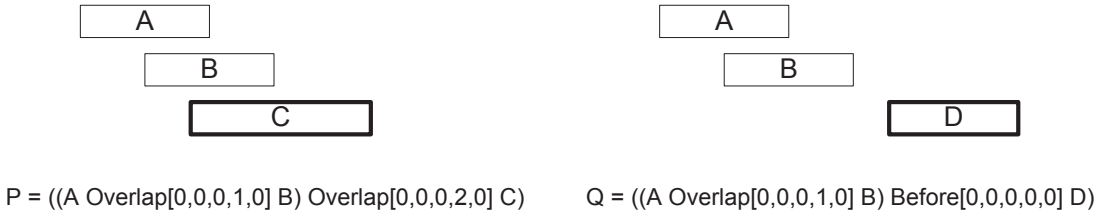


Figure 3.9. Candidate generation: joining frequent pattern P to frequent pattern Q

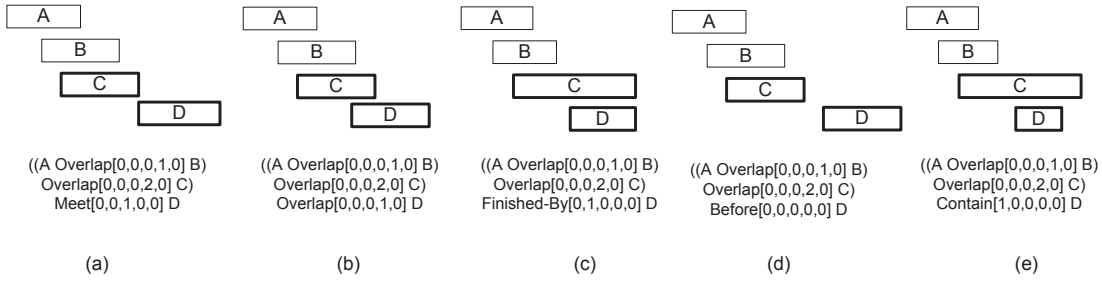


Figure 3.10. Generated candidate patterns from two frequent patterns given in Figure 3.9.

Based on this observation, we introduce the concept of a **dominant** event in a temporal pattern. A dominant event in the pattern P is an event from P with the latest start time. During the candidate generation process, a frequent $(k-1)$-pattern P is joined to frequent $(k-1)$-pattern

Q if prefix of P is equal to prefix of Q. This joining generate candidate k-pattern R such that the temporal relationship between dominant events of P and Q in candidate pattern R is frequent.

We explain the proposed candidate generation process as follow. Consider the set of frequent 3-patterns, $freSet_3$, given in Table 3.3(a) and the set of frequent 2-patterns, $freSet_2$, given in Table 3.3(b). The dominant event of the frequent 3-pattern is underlined in Table 3.3(a). Our purpose is to generate a set of 4-patterns, $canSet_4$. To generate the set of candidate 4-patterns, we use two 3-patterns from $freSet_3$, such that they share common prefix. For example, consider joining P to Q, where P = (A Overlap[0,0,0,1,0] B) Overlap[0,0,0,2,0] $\underline{C}$ and Q = (A Overlap[0,0,0,1,0] B) Before[0,0,0,0,0] $\underline{D}$. Event C is dominant event in P and event D is dominant event in Q. The frequent temporal relation between dominant events C and D in $freSet_2$ are: $C\ Contain[1,0,0,0,0]\ D$ and $C\ Before[0,0,0,0,0]\ D$. Thus, the generated patterns are

$((A\ Overlap[0,0,0,1,0]\ B)\ Overlap[0,0,0,2,0]\ C)\ Contain[1,0,0,0,0]\ D$, and

$((A\ Overlap[0,0,0,1,0]\ B)\ Overlap[0,0,0,2,0]\ C)\ Before[0,0,0,0,0]\ D$.

Similarly, consider joining P' to Q', where P' = (A Overlap[0,0,0,1,0] B) Before[0,0,0,0,0] $\underline{D}$ and Q' = (A Overlap[0,0,0,1,0] B) Before[0,0,0,0,0] $\underline{F}$. The frequent temporal relation between event D and F in $freSet_2$ is $D\ Before[0,0,0,0,0]\ F$. Thus, the generated pattern is (($A\ Overlap[0,0,0,1,0]\ B$) $Before[0,0,0,0,0]\ D$) $Before[0,0,0,0,0]\ F$. Table 3.4 presents the list of generated 4-patterns. Later, pattern from this set will be pruned if it does not satisfy downward closure property.

One key point to note that not all frequent 2-patterns from $freSet_2$ are useful during candidate generation. For example, consider frequent 2-pattern $C\ Before[0,0,0,0,0]\ D$ and assume, we are generating a set of 4-patterns from the set of frequent 3-patterns. Careful investigation reveals that, no pattern from $freSet_3$ contains $C\ Before[0,0,0,0,0]\ D$. In other words, no pattern

(a) $freSet_3$

A Overlap[0,0,0,1,0] B Before[0,0,0,0,0] D
A Overlap[0,0,0,1,0] B Overlap[0,0,0,2,0] C
A Overlap[0,0,0,1,0] B Before[0,0,0,0,0] F
A Before[0,0,0,0,0] D Before[0,0,0,0,0] F
A Overlap[0,0,0,1,0] C Contain[1,0,0,0,0] D
A Before[0,0,0,0,0] F Before[0,0,0,0,0] G
B Overlap[0,0,0,1,0] C Contain[1,0,0,0,0] D
B Before[0,0,0,0,0] D Before[0,0,0,0,0] F

(b) $freSet_2$

A Overlap[0,0,0,1,0] B
A Before[0,0,0,0,0] D
A Overlap[0,0,0,1,0] C
A Before[0,0,0,0,0] F
A Before[0,0,0,0,0] G
B Overlap[0,0,0,1,0] C
B Before[0,0,0,0,0] D
B Before[0,0,0,0,0] F
C Contain[1,0,0,0,0] D
C Before[0,0,0,0,0] D
D Before[0,0,0,0,0] F
F Before[0,0,0,0,0] G

Table 3.3. Generating 4-patterns

3-patterns
(A Overlap[0,0,0,1,0] B) Overlap[0,0,0,2,0] C Before[0,0,0,0,0] D
(A Overlap[0,0,0,1,0] B) Overlap[0,0,0,2,0] C Contain[1,0,0,0,0] D
(A Overlap[0,0,0,1,0] B) Before[0,0,0,0,0] D Before[0,0,0,0,0] F

Table 3.4. Intermediate $canSet_4$

of length 3 that has $C\ Before[0,0,0,0,0]\ D$ is frequent. According to anti-monotone property, any pattern of length 4 that has $C\ Before[0,0,0,0,0]\ D$ will not be frequent. Thus, frequent 2-pattern $C\ Before[0,0,0,0,0]\ D$ cannot be used to generate candidate pattern with length ≥ 4.

Property II *Suppose a k-pattern R is generated by joining frequent $(k-1)$-pattern P to frequent $(k-1)$-pattern Q. Let TP_2 be a temporal pattern between events E_i and E_j in pattern R, where event E_i is from P and event E_j is from Q. Pattern R cannot be frequent if TP_2 is present in less than $(k-2)$ number of frequent $(k-1)$-patterns.*

Proof We prove by contradiction. Assume a k-pattern R is frequent and pattern TP_2 is present in less than $(k-2)$ number of frequent $(k-1)$-patterns. By the downward closure property, a frequent k-pattern R has k subpatterns of length $k-1$. Among them, we have at least $(k-1)$ subpatterns

that contain event E_i. Similarly, we have at least $(k-1)$ subpatterns that contain event E_j. This implies that both E_i and E_j must occur in at least $(k-2)$ number of frequent subpatterns of length $(k-1)$. In other word, TP_2 must occur in at least $(k-2)$ number of frequent subpatterns of length $(k-1)$. □

Based on the above observation, while generating $canSet_k$ from $freSet_{k-1}$, we use only those 2-patterns from $freSet_2$ that are present in at least $(k-2)$ number of frequent patterns from $freSet_{k-1}$.

To illustrate working of above observation, consider a case where 2-pattern is present in less than $k-2$ number of frequent $(k-1)$-patterns and utilized for candidate generation process. One such pattern is $C\ Before[0,0,0,0,0]\ D$ while generating $canSet_4$ using $freSet_3$. Recall, we have used $C\ Before[0,0,0,0,0]\ D$ to generate 4-pattern P = ((A Overlap[0,0,0,1,0] B) Overlap[0,0,0,2,0] C) Before[0,0,0,0,0] D. According to downward closure property all length 3 subpatterns of P must be frequent. However, subpatterns "(A Overlap[0,0,0,1,0] C) Before[0,0,0,0,0] D" and "(B Overlap[0,0,0,1,0] C) Before[0,0,0,0,0] D" are not frequent. Hence, even if we utilize such 2-pattern which are present in less than 2 frequent patterns in $freSet_3$, we are not generating any temporal pattern that can be frequent. Next, we show that our algorithm mines complete set of frequent patterns.

Theorem 1 *Algorithm IEMiner is complete.*

Proof Initially, Algorithm IEMiner generates all frequent 2-patterns. In the subsequent iterations, IEMiner generates k-pattern from two frequent $(k-1)$-patterns and frequent 2-pattern. Now, assume that IEMiner generates all frequent $(k-1)$-patterns. We prove that algorithm IEMiner will generate all frequent k-patterns.

Suppose IEMiner does not generate all frequent k-patterns, in other words, there exists a frequent k-pattern R that has not been generated by our algorithm. Without loss of generality, suppose R can be generated from two frequent $(k-1)$-patterns P and Q. Let 2-pattern TP_2 denote the temporal relationship between dominant events of P and Q in pattern R. Here, patterns P and Q are generated by IEMiner. The only way in which k-pattern R is not generated is if TP_2 is missing from the set of 2-patterns used during candidate generation.

As k-pattern R is frequent, this implies that there are k number of frequent $(k-1)$-subpatterns of R. Among these k frequent subpatterns, TP_2 must be present in at least $k-2$ patterns. By Property II, TP_2 will be generated and used during candidate generation, indicating that k-pattern R will be generated by IEMiner. This completes the proof.□

3.3.2 Support Counting

After the $canSet_k$ has been generated, we need to count the number of occurrences of each k-pattern to determine whether they are frequent or not. Traditionally, support counting is done by scanning the event list for each candidate pattern. However, checking the occurrence of a k-pattern in a given event list with m events takes O(km) time. Repeating this process for n k-patterns takes O(kmn) time. In other words, an event in the event list will end up being scanned multiple times.

Instead, we utilize a single-pass support counting procedure where each event in the event list is scanned only once to determine the occurrence of all k-patterns. Algorithm CountSupport (see Algorithm 3) gives the details. The inputs are an event list EL, a set of k-patterns $canSet_k$, and the level number L(i.e., length of temporal pattern). The idea is to keep track of the active events as we scan the event list. An event is considered active at time point t if the start time of the event is

less than t while the end time of the event is greater than t. Otherwise, the event is passive. Active events are maintained in $active\_TP$ list and passive events are maintained in $passive\_TP$ list. As a new event E arrives, we update the $active\_TP$ and the $passive\_TP$ to reflect the completion of previously active events (Lines 7-10). Next, new composite events are formed between events from the $active\_TP$ and the new event E (Lines 11-12). If the composite event is present in $canSet_k$, its support count is incremented (Lines 13-14). If it is the prefix of any pattern from $canSet_k$, we store it in the $active\_TP$ (Lines 15-17). Similarly, new composite events are formed between events from the $passive\_TP$ and the new event E (Lines 20-28). Event E is then inserted into $active\_TP$ (Line 29). With this, we only need to scan the event list once and count the support of all candidate patterns.

To illustrate the support counting process, let us consider the two patterns (A Overlap[0,0,0,1,0] B) Overlap[0,0,0,2,0] C and (A Before[0,0,0,0,0] C) Overlap [0,0,0,1,0] D in $canSet_3$ and the first event list from Table 3.1. Table 3.5 shows the patterns generated as we process an active event. The patterns in italic are discarded since they are not the prefix of any patterns in $canSet_3$. The pattern (A Overlap[0,0,0,1,0] B) Overlap[0,0,0,2,0] C is the only pattern present in first event list and we increase its support count.

$nextEvent$	Generated pattern	$passive\_TP$	$active\_TP$
A	-	-	A
B	(A Overlap[0,0,0,1,0] B)	-	A, B (A Overlap[0,0,0,1,0] B)
C	**(A Overlap[0,0,0,1,0] B) Overlap[0,0,0,2,0] C** *A Overlap[0,0,0,1,0] C* *B Overlap[0,0,0,1,0] C*		A, B, C
D	*C Contain[1,0,0,0,0] D* *A Before[0,0,0,0,0] D* *B Before[0,0,0,0,0] D*	A, B	C, D

Table 3.5. Support counting of $canSet_3$ using first event list from Table 3.1

Algorithm 3 Algorithm countSupport

Input: Level L, Event List EL, $canSet_k$
Output: $canSet_k$ with updated count

```
1: active_TP ← φ
2: passive_TP ← φ
3: while ((nextEvent ← getNextEvent(EL)) ≠ NULL ) do
4:    if (nextEvent is frequent event) then
5:       currentTime = nextEvent.start
6:       for all (temporal pattern tp ∈ active_TP) do
7:          if (tp.end < currentTime) then
8:             passive_TP ← passive_TP ∪ tp
9:             active_TP ← active_TP − tp
10:         else
11:            relation ← getRelation(tp, nextEvent)
12:            newTP = prepareNewTP(tp, nextEvent, relation)
13:            if (newTP.size = L && newTP ∈ canSet_k) then
14:               Update count for newTP in canSet_k
15:            else if (newTP is a prefix of a pattern in canSet_k) then
16:               active_TP ← active_TP ∪ newTP
17:            end if
18:         end if
19:      end for
20:      for all (tp ∈ passive_TP) do
21:         relation ← Before
22:         newTP = prepareNewTP(tp, nextEvent, relation)
23:         if (newTP.size = L && newTP ∈ canSet_k) then
24:            Update count for newTP in canSet_k
25:         else if (newTP is a prefix of a pattern in canSet_k) then
26:            active_TP ← active_TP ∪ newTP
27:         end if
28:      end for
29:      active_TP ← active_TP ∪ nextEvent
30:   end if
31: end while
32: return canSet_k
```

3.3.3 Optimization Strategies

Besides the novel candidate generation and support counting procedures, we further introduce two optimization strategies to achieve greater efficiency for IEMiner.

The first strategy involves building a list of *blacklisted* event list. An event list EL is *blacklisted* if it has less than k frequent events as such EL does not have enough events to generate a k-pattern and hence it cannot affect the support counts of the k-patterns. This implies we can safely omit EL from scanning during the support counting procedure from the k^{th} iteration onwards.

The second optimization strategy aims to further reduce the number of candidate patterns generated by utilizing the following observation.

Property III: (Prefix Count) Suppose a k-pattern R is generated by joining frequent $(k-1)$-pattern P to frequent $(k-1)$-pattern Q. Let $w1$ denote the number of event lists in which the prefix of P occurs at least twice and $w2$ denote the number of event lists in which the prefix of Q occurs at least twice. Pattern R cannot be frequent if $w1 < minsup$ or $w2 < minsup$.

Proof We prove this by contradiction. Assume k-pattern R is frequent and $w1 < minsup$ or $w2 < minsup$. As patterns P and Q are subpatterns of R, they occur together in at least $minsup$ number of event lists. Also, pattern R is generated from P and Q. Thus, P and Q have same prefix. In other word, there are at least $minsup$ number of event lists in which prefix of P and prefix of Q occurs twice. Thus $w1 >= minsup$ and $w2 >= minsup$. This is a contradiction. □

With this observation, we maintain the w values for each candidate k-pattern R as we scan the event list during support counting. The w value is incremented if there is another temporal pattern having the same prefix of R being generated in same event list and the window has not been blacklisted. Apriori-based candidate generation process use all frequent temporal patterns

discovered at the current round to generate candidates pattern for the next round. However, using the above observation, we only need to join those frequent patterns whose $w >= minsup$.

3.4 Algorithm IEClassifier

To the best of our knowledge, this is the first work on utilizing interval-based temporal patterns for classification. Existing works utilized frequent itemset patterns for classification. However, the direct adaptation of existing approaches for the interval-based patterns is not straightforward and scalable due to the large number of frequent temporal patterns generated. On the other hand, transforming the temporal patterns by treating each temporal relation between any two events as an independent attribute will result in a very high dimensional space and may suffer from the curse of dimensionality. To address these problems, we propose a classifier called IEClassifier(Interval based Event Classifier).

The building of IEClassifier has two aspects. The first aspect deals with the selection of a subset of patterns that is able to discriminate one class from another with high degree of accuracy. The second aspect deals with the assignment of an unknown input event sequence to a class given the selected subset of patterns.

Frequent interval-based temporal patterns are generated from a set of training data that has been partitioned according to their class labels C_i, $1 \leq i \leq c$, where c is the number of class labels. A frequent pattern which occurs in only one class is more discriminating than one that occurs in all the classes. To identifying such discriminating patterns, we compute the information gain of each pattern TP using the following equation:

$$\begin{aligned}\text{InfoGain}(TP)= \quad & -\sum_{i=1}^{c} p(C_i) log\ p(C_i) + \\ & p(TP) \sum_{i=1}^{c} p(C_i|TP) log\ p(C_i|TP) + \\ & p(\overline{TP}) \sum_{i=1}^{c} p(C_i|\overline{TP}) log\ p(C_i|\overline{TP})\end{aligned}$$

In the above formula, $p(TP)$ is probability of pattern TP to occur in datasets. Also $p(\overline{TP})$=1-$p(TP)$. We calculate information gain for all frequent patterns using above formula. Those temporal patterns whose information gain values are below a predefined $info\_gain$ threshold are removed. The remaining temporal patterns are the discriminating patterns. We assign to each discriminating pattern the class label with the highest conditional probability $p(C|TP)$. $p(C|TP)$ is also known as the confidence of TP (conf(TP)). At the end of the process, each discriminating pattern TP is assigned a class label (clabel(TP)) with the support count sup(TP).

Algorithm 4 Algorithm Best_Conf

Input: Event sequence I, $PatternMatch_I$
Output: Class Label of I

```
1: best_class ← Default class label
2: conf ← 0
3: sup ← 0
4: for all (temporal pattern TP ∈ PatternMatch_I) do
5:    if (conf(TP) > conf) then
6:       best_class = class label of TP
7:       conf = conf(TP)
8:       sup = sup(TP)
9:    else if (conf(TP) == conf and sup(TP) > sup) then
10:      best_class = class label of TP
11:      conf = conf(TP)
12:      sup = sup(TP)
13:   end if
14: end for
15: Assign best_class as a class label to I
```

For an unknown input event list I, we match I against all the discriminating patterns. Let $PatternMatch_I$ be the set of discriminating patterns that are contained in I. Intuitively, there are

Algorithm 5 Algorithm Majority_Class

Input: Event sequence I, $PatternMatch_I$
Output: Class Label of I

```
1: for all (class C_i, 1 ≤ i ≤ c) do
2:    count[C_i] ← 0
3: end for
4: for all (TP ∈ PatternMatch_I) do
5:    clabel ← class label of TP
6:    count[clabel]++
7: end for
8: max ← 1
9: for all class C_i, 1 ≤ i ≤ c do
10:   if (count[C_i] > count[max]) then
11:      max ← i
12:   end if
13: end for
14: Assign class C_max to I
```

two ways to assign a class label to I. The first way is to assign I to the class label with the highest confidence pattern in $PatternMatch_I$. The second way is to assign I to the majority class labels of the patterns in $PatternMatch_I$. Algorithms Best_Conf (see Algorithm 4) and Majority_Class (see Algorithm 5) show the details.

3.5 Empirical Studies

In this section, we present the results of experiments conducted to evaluate IEMiner and IEClassifier.

We first compare the performance of IEMiner with state-of-the-art algorithms GenPrefixSpan [14], TPrefixspan [93] and H-DFS [72] to evaluate its efficiency and scalability. We use GenPrefixSpan as the baseline for IEMiner since it only finds the Before relationship while IEMiner is able to generate all the temporal relationships among the events. Then we examine the effectiveness of the two optimization strategies. We also apply IEMiner on three real world datasets, namely

the American Sign Language (ASL) dataset[4], Stulong dataset[5] and the Hepatitis dataset[6]. Finally, we verify the accuracy of IEClassifier on the Hepatitis and Stulong data sets.

All the algorithms are implemented in C#. The experiments are performed on a 1.6 GHz centrino duo with 1.5GHz RAM running window operating system. We modify the IBM data quest generator[7] by including an additional parameter "$EvtDen$"(i.e., number of events active at a time) to generate the synthetic data sets. The control parameters used in the data generator are:

1. number of windows (i.e., D)
2. number of event types (i.e., T)
3. average number of events, active at a time (i.e., $EvtDen$)
4. average length of patterns (i.e., L)
5. probability of similar event appear in same window (i.e., P)

We keep T = 500 for all the experiments. The notation "$Data\_D\_T\_L\_P\_EvtDen$" represents dataset generated using D, T, L, P and $EvtDen$ control parameters.

3.5.1 Experiments on Synthetic Datasets

First, we analyze the effect of varying minimum support on runtime. Figure 3.11 shows the results when minimum support varies from 2% to 12%. We observe that as support value decreases, the time required by all the algorithms increases. However, the runtime for H-DFS and TPrefixSpan increase drastically compared to IEMiner. We also note that IEMiner has a comparable

[4]http://www.bu.edu/asllrp/
[5]http://ecmlpkdd.isti.cnr.it/
[6]http://ecmlpkdd.isti.cnr.it/
[7]http://www.almaden.ibm.com/software/quest/Resources/ index.shtml

runtime as GenPrefixSpan even though GenPrefixSpan only finds the Before relationship while IEMiner generates all types of interval-based relationships.

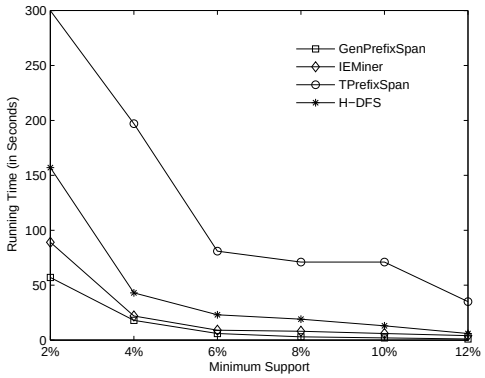


Figure 3.11. Effect of Varying Minimum Support

Next, we examine the effect of varying sizes of D on runtime. We select 4% as a support value and vary D from 100K windows to 400K windows. Average number of events in each window is 15, hence average number of events vary from 1500K to 6000K. Figure 3.12 shows the experimental results. The runtime of IEMiner increases linearly as value of D increases while the runtime of TPrefixSpan increases exponentially.

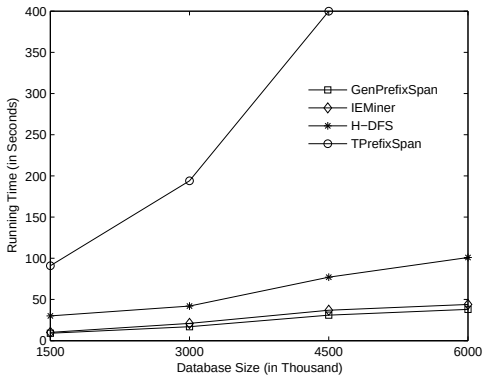


Figure 3.12. Effect of Varying Database Size (Data_?k_500_15_0.3_2)

We also investigate the effect of varying L on run time. We keep D and the support value constant. Figure 3.13 shows the results. As the value of L increases, the runtime of IEMiner increases but at a slower rate compared to H-DFS and TPrefixSpan. This demonstrates that IEMiner is effective in reducing the number of candidates generated, thereby allowing a much longer pattern to be discovered.

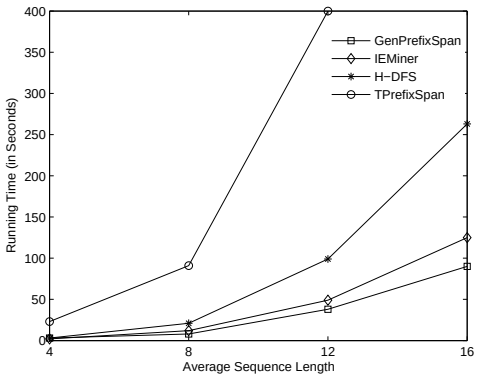


Figure 3.13. Effect of Varying Pattern Length (Data_200k_500_?_0.3_2)

In the next set of experiments, we investigate the effect of varying $EvtDen$ on run time. Figure 3.14 shows the runtime of IEMiner for varying values of $EvtDen$. We observe that as $EvtDen$ increases, the number of temporal relations among the events also increases. Hence, the support count for each pattern is reduced. As a result, fewer number of frequent patterns are generated compared to GenPrefixSpan. Note that $EvtDen$ = 1 means that there is only one active event at each time.

Finally, we analyze the effectiveness of the two optimization strategies. Two variations of IEMiner are implemented. IEMiner-1 uses only the event list blacklisting optimization strategy, while IEMiner-2 uses only the prefix count optimization strategy. Figure 3.15 shows the results.

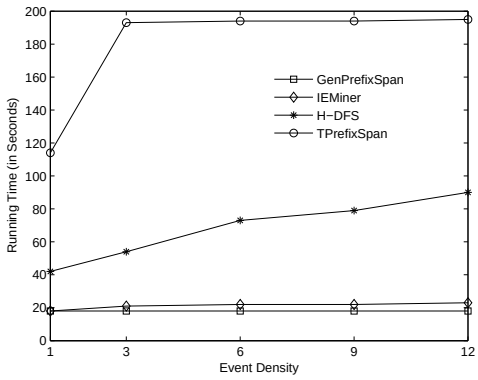


Figure 3.14. Effect of Varying Event Density (minimum support = 4%)

We see that the window blacklisting strategy (IEMiner-1) is able to improve the performance of IEMiner more as compared to the prefix count strategy (IEMiner-2).

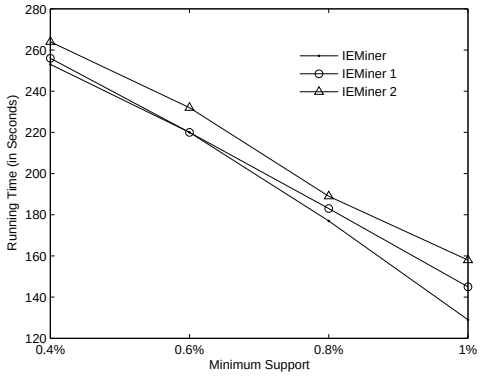


Figure 3.15. Effect of Optimization Techniques (Data_200k_500_20_0.3_2)

3.5.2 Experiments on Real World Datasets

In this section, we apply the four mining algorithms (IEMiner, TPrefixSpan, H-DFS and GenPrefixSpan) on three real world datasets, namely, the American Sign Language (ASL) dataset, Stulong dataset and the Hepatitis dataset.

ASL dataset

We use the ASL dataset to investigate the relationship between grammatical structure and gesture field. This dataset has 730 utterances. Each utterance contains recurrent ASL gestural and grammatical field. We obtain the frequent temporal patterns at various support values. The set of mined patterns is verified against the ground truth [72]. The results are shown in Figure 3.16.

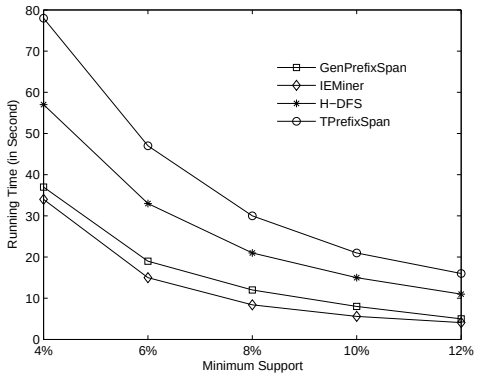


Figure 3.16. Experiments on ASL dataset

Hepatitis dataset

The Hepatitis dataset contains a total of 771 patient records over a period of 10 years. In this dataset, a patient either has Hepatitis B or Hepatitis C. There are about 230 tests that a patient may undergo, out of which 25 tests are conducted regularly at each visit to the hospital. We transform the test results over time into interval based events as follows:

1. If the results of a test $Test$ during an interval $[start, end]$ consistently falls within the normal range of values for the test $Test$, that is, N(ormal), we map it to the event ($Test$-N, $start, end$).

2. If the results of a test $Test$ during an interval $[start, end]$ consistently falls below the normal range of values for the test $Test$, that is, L(ow), we map it to the event ($Test$-L, $start, end$).

3. If the results of a test $Test$ during an interval $[start, end]$ consistently falls above the normal range of values for the test $Test$, that is, H(igh), we map it to the event ($Test$-H, $start, end$).

4. If the results of a test $Test$ during an interval $[start, end]$ oscillates between Low and Normal, we map it to the event ($Test$-NL, $start, end$).

5. If the results of a test $Test$ during an interval $[start, end]$ oscillates between Normal and High, we map it to the event ($Test$-NH, $start, end$).

6. If the results of a test $Test$ during an interval $[start, end]$ oscillates between Low and High, we map it to the event ($Test$-LH, $start, end$).

After mapping the test results into interval based events, we create an event list for each patient. We obtain a total of 498 event lists that correspond to patients who undergo the 25 tests regularly.

Figure 3.17 shows the results of applying the mining algorithms on the transformed Hepatitis dataset (Hep-T). We observe that IEMiner perform best compared to all algorithms. Here, average length of underlying event list is around 200 events. GenPrefixspan did not perform well because it consider events without duration and as a result many patterns are generated compared to other three algorithm.

Stulong dataset

The Stulong dataset contains a total of 860 patient records over a period of 20 years. In this dataset, a patient either has Cardiovascular disease or not. There are about 10 tests that a patient

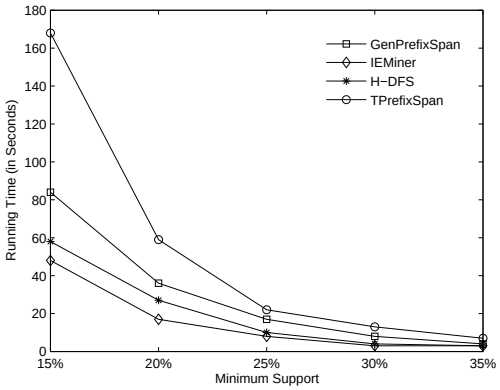


Figure 3.17. Experiments on Hepatitis dataset

may undergo at each visit to the hospital. We transform the test results over time into interval based events by following similar convection as explained for hepatitis data. Figure 3.18 shows the results of applying the mining algorithms on the transformed Stulong dataset. We observe that IEMiner performs best compared to all algorithms.

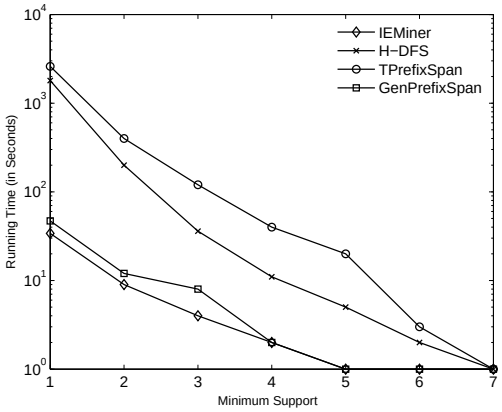


Figure 3.18. Experiments on Stulong dataset

3.5.3 Accuracy of IEClassifier

Finally, we investigate whether the discovered temporal patterns will improve the accuracy of classification. We compare the accuracy of IEClassifier with standard classifiers such as C4.5, CBA and SVM which do not use the temporal information.

Hepatitis dataset

We apply C4.5[8], CBA[9] and SVM[10] classification tools on the original Hepatitis dataset where each test per visit is considered as an attribute. In total, we have around 10,000 attributes.

Next, we build the IEClassifier from the interval-based Hep-T dataset obtained in the previous section. We label an event list in the Hep-T dataset as HepB or HepC to indicate that the patient corresponding to the event list has Hepatitis B or Hepatitis C. In total, we have 203 event lists labeled as HepB and 295 event lists labeled as HepC. The info-gain threshold is set at 0.02 with minsup of 10%.

10-fold cross validation testing strategy is adopted. Table 3.6 shows the results.We observe that the classifiers that make use of temporal relationships can indeed improve the prediction accuracy. Overall, the Majority_Class voting strategy achieves the best accuracy.

Classifier	Testing Accuracy
C4.5	78.13%
CBA	76.49%
SVM	78.72%
IEMiner (Majority_Class)	**82.13**%
IEMiner (Best_Conf)	78.91%

Table 3.6. Testing accuracy: Hepatitis Dataset

[8]http://www.cs.waikato.ac.nz/ml/weka/
[9]http://www.comp.nus.edu.sg/ dm2/
[10]http://svmlight.joachims.org/ and also weka

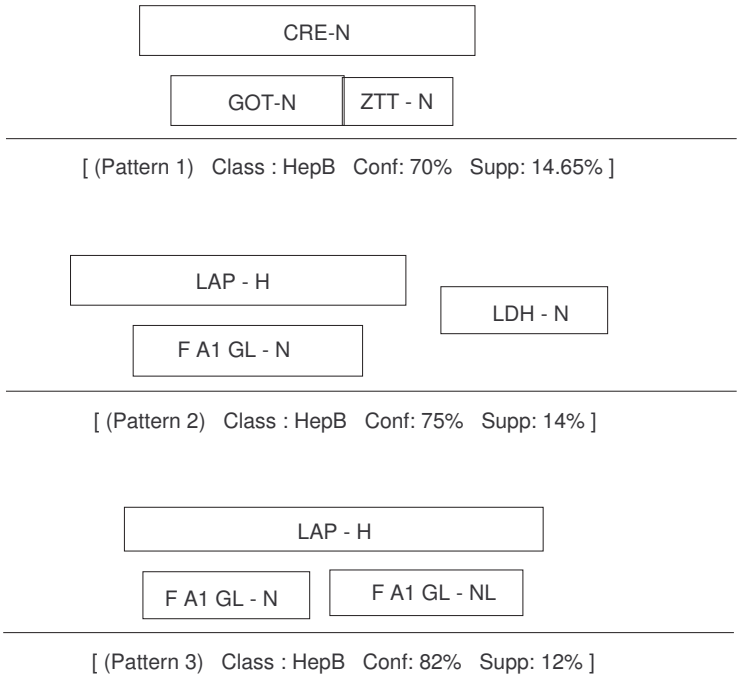


Figure 3.19. Sample of temporal patterns for Hepatitis B disease

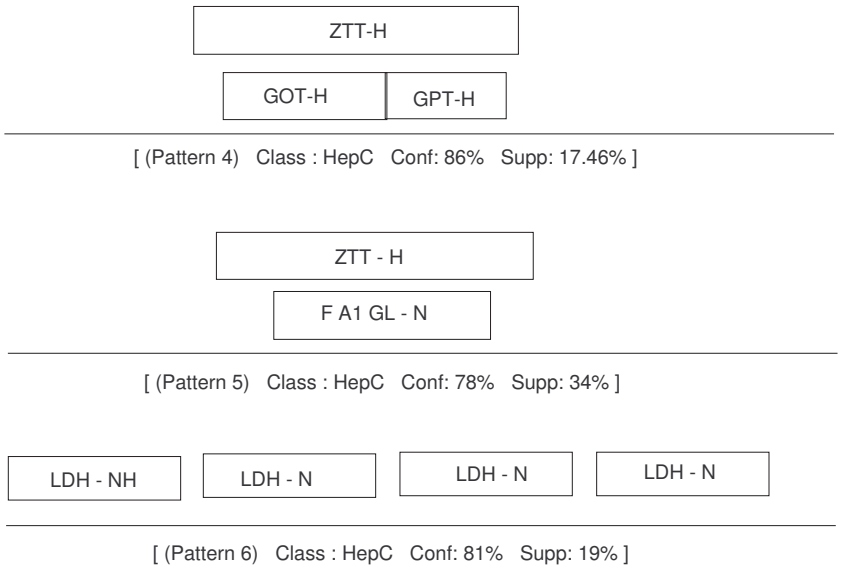


Figure 3.20. Sample of temporal patterns for Hepatitis C disease

Figure 3.19 and Figure 3.20 show a sample of the temporal patterns that is able to discriminate between the HepB and HepC classes. The first 5 patterns reveal the temporal relations between different tests in the Hepatitis B and Hepatitis C patients. For example, pattern 3 describes the behavior of F-A1.GL with respect to LAP test. We discover that during the period in which LAP's value is in the normal range, the F-A1.GL test starts in the normal range and then begins to oscillate between the low and normal range. This pattern is observed in Hepatitis B patients with 82% confidence. It is present in 25.54% hepatitis B patient data and 3% of hepatitis C patient data. Pattern 6 reveals how a particular test, the LDH test, evolves in the Hepatitis C patients. Initially, the LDH's value ranges between normal and high, and as time passes, it's value becomes normal. This pattern is present in 26% of hepatitis C patient as opposed to 8% in hepatitis B patients.

Stulong dataset

We apply C4.5[11], CBA[12] and SVM[13] classification tools on the original Stulong dataset where each test per visit is considered as an attribute. In total, we have around 700 attributes.

Next, we build the IEClassifier from the interval-based Stulong dataset obtained in the previous section. We label an event list in the Stulong dataset as CVD or No-CVD to indicate that the patient corresponding to the event list has cardiovascular disease or not. In total, we have 460 event lists labeled as No-CVD and 295 event lists labeled as CVD. The info-gain threshold is set at 0.02 with minsup of 10%.

10-fold cross validation testing strategy is adopted. Table 3.7 shows the results. We observe that the classifiers that make use of temporal relationships can indeed improve the prediction

[11] http://www.cs.waikato.ac.nz/ml/weka/
[12] http://www.comp.nus.edu.sg/ dm2/
[13] http://svmlight.joachims.org/ and also weka

Classifier	Testing Accuracy
C4.5	69.78%
CBA	69.56%
SVM	70.43%
IEMiner (Majority_Class)	**76.13**%
IEMiner (Best_Conf)	71.91%

Table 3.7. Testing accuracy: Stulong Dataset

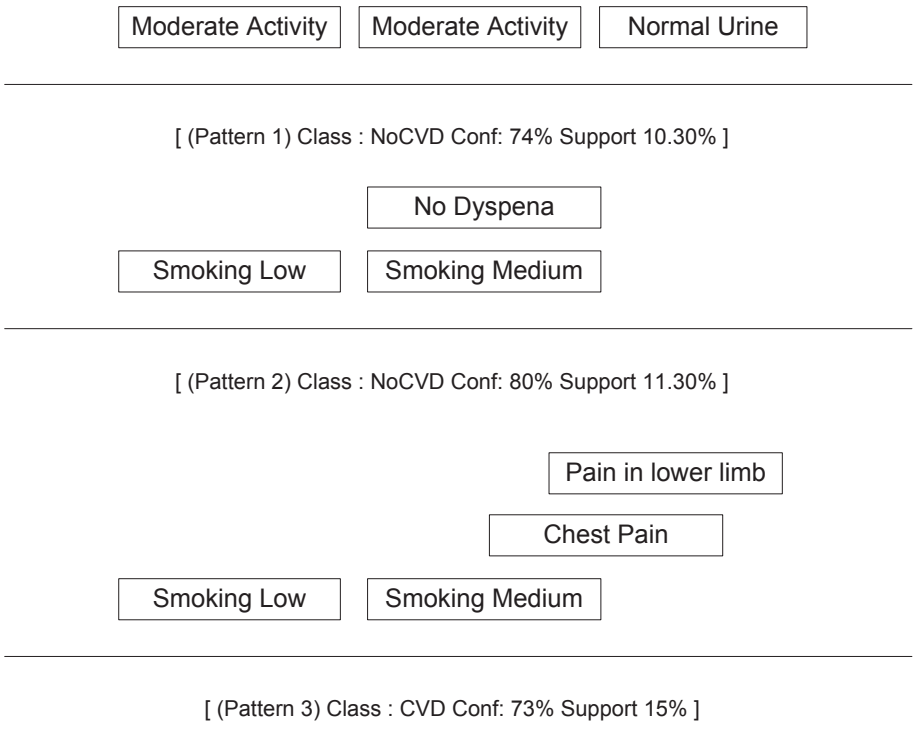


Figure 3.21. Sample of temporal patterns for Stulong dataset

accuracy. Overall, the Majority_Class voting strategy achieves the best accuracy. Figure 3.21 shows a sample of the temporal patterns that is able to discriminate between CVD and No-CVD classes. It was assumed that, smoking is a big risk factor for cardiovascular disease. But, from patterns 2 and 3 in Figure 3.21, we can see, a smoker with chest pain and pain in lower limb will develop CVD in future.

3.6 Summary

In this chapter, we have presented a novel approach to mine temporal patterns from interval data. Our key contributions are as follow:

1. We augment the hierarchical representation with count information to achieve a lossless representation. We provide a proof that the augmented representation is indeed lossless. This enables us to recover the actual relationships among events from the temporal pattern.

2. We design an Apriori-based algorithm called "IEMiner"(**I**nterval-based **E**vent **Miner**) to discover frequent temporal patterns based on the lossless representation. IEMiner employs two optimization strategies to reduce the search space. The proof of the completeness of IEMiner is detailed.

3. We also build an interval-based temporal pattern classifier called IEClassifier to perform the classification of closely related classes. We apply the classifier to a real world Hepatitis dataset and Stulong dataset to demonstrate its improved accuracy.

The success of our approach on all the tested real-world datasets indicates that the event duration play an important role in extracting complex relationship among durative events. Our

approach can easily incorporate user's intension in lossless representation definition. Further, the proposed algorithm can easy work with point events and durative events. In the future, we would like to consider mining temporal pattern from uncertain durative events. Also, it would be interesting to evaluate the viability of the proposed approach in a streaming or incremental setting.

Chapter 4

Mining Patterns from Time Series Data

In the previous chapter, we have analyzed a set of event sequences, where each event sequence is a collection of interval-based events. In this chapter, we analyze the set of time series, where each time series is a sequence of real valued observations. We have seen that the recent research interests in time series data mining mainly involve indexing time series for efficient similarity search, clustering time series, motif discovery, rule discovery, time series correlation and so on. Time series motif discovery is an active research topic [25, 50, 63, 64]. Time series motifs are the recurring patterns in single time series. For example, Figure 4.1 shows the sample time series and one motif m = $\{s_1, s_3, s_3, s_4\}$. The length of motif m is 10 and it appears 4 times in time series. Attempts have been made to generalize the notion of motifs from single time series to multi-dimensional time series data [99, 57, 68, 88]. This generalization allows the handling of real world applications involving several data sources such as activity discovery using wearable sensor data, gene expression data showing the expression levels of multiple genes, stock market data giving the

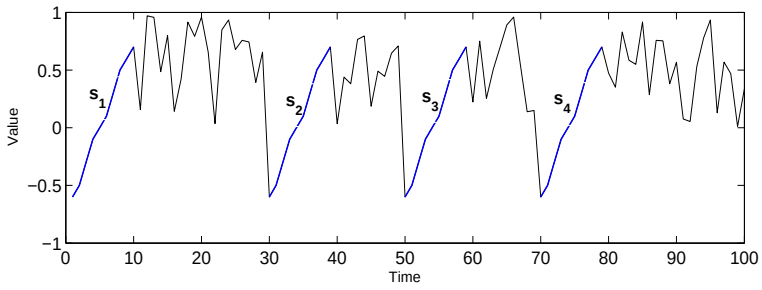


Figure 4.1. Time series motif.

stock prices of diverse companies. However, none of these methods considers the ordering among the motifs in such an environment.

Figure 4.2 shows the time series of QLogic, Intel and JP Morgan stocks. Motifs m_1 = $\{s_{11}, s_{12}, s_{13}\}$, m_2 = $\{s_{21}, s_{22}, s_{23}\}$ and m_3 = $\{s_{31}, s_{32}, s_{33}\}$ are highlighted in the time series of QLogic, Intel and JP Morgan stocks respectively. A closer examination of the motifs in Figure 4.2 reveals that the subsequences from one motif occurs at a consistent lag relative to subsequences from other motifs. For example, s_{21} occurs with lag 6 relative to s_{11} while s_{31} occurs with lag 7 relative to s_{11}. This pattern is repeated for (s_{12}, s_{22}, s_{32}) and (s_{13}, s_{23}, s_{33}). In short, the lag relationship among the subsequences are *repeated*. The existence of such invariant ordering among the motifs suggests that there may exist some hidden relationships. Further investigation[1] reveals that QLogic stock is competitor of Intel stock, while JP Morgan stock gives higher rating for investment in Intel Stock.

The existence of such invariant orderings among time series motifs is useful and may provide critical insights in many time series applications. For example:

- **Financial Domain**: Existing approaches compute the portfolio's expected risks based on the co-variances among the assets time series in the portfolios. Alternatively, we can model the

[1] Yahoo Finance - http://finance.yahoo.com

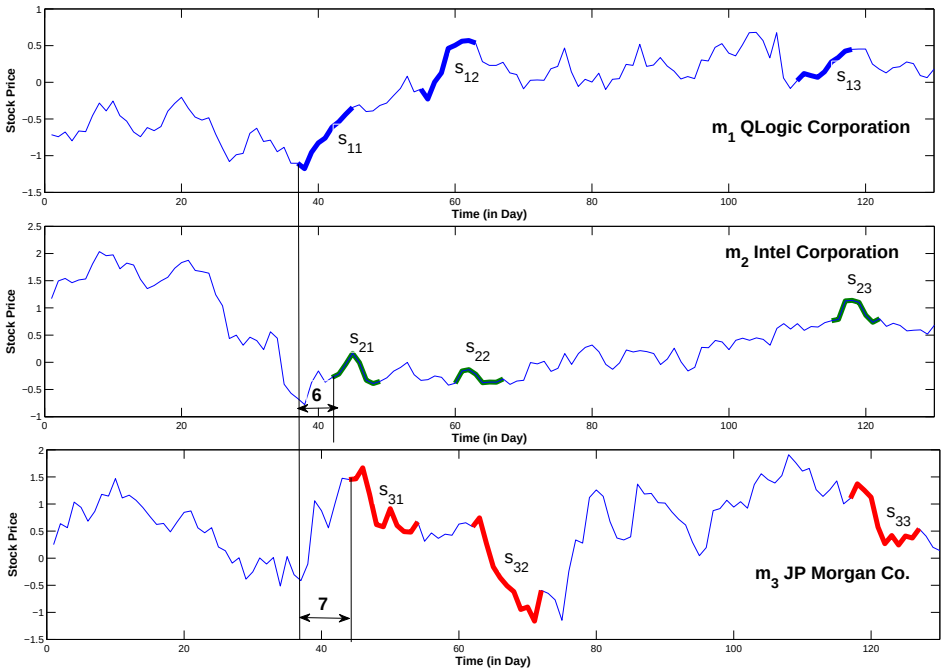


Figure 4.2. Lag relationships among motifs m_1, m_2 and m_3 reflecting competitor/co-operative behavior.

portfolio's risks by considering the competitor or co-operative relationship between the assets time series in the portfolio. For example, given the performance of two financial assets, A and B, where we know that whenever the price of A drops, the price of B will drop in next five days. Intuitively, it may not be appropriate to construct a portfolio by including both A and B concurrently, as the exposure of loss will be increased. Yet, such kind of relationship cannot always be captured by co-variances. We consider dependence relationship among assets time series and construct the portfolio. Our experiments reveal that stock portfolio based on lag relationships leads to increase in the cumulative rate of return on investment.

- **Medical Domain**: The electrocardiogram (ECG) are standard test for diagnosing various cardiovascular abnormalities. Typically, various sensors are placed in different parts of the body. Physicians monitor the various channels of signals to recognize warning patterns. By auto-

matically discovering invariant orderings among multi-dimensional motifs, they may serve as early warning patterns to allow for timely intervention.

In this chapter, we introduce the notion of ***lagPatterns*** in time series data to capture the orderings among motifs from different time series. Unlike existing multi-dimensional motifs, *lagPattern* explicitly accounts for lags and the ordering among the motifs from different time series. Finding *lagPattern* involves two main steps:

1. Identify all motifs of various length in single time series.

2. Discover groups of motifs from multiple time series with invariant orderings.

Both steps are computationally expensive. A time series of length L, without discretization, would have $O(L^2)$ subsequences of various length and hence $O(L^2)$ motifs. Thus, the naive enumeration based method for the first step is quadratic. With N time series, we would have $O(L^{2N})$ possible *lagPatterns*. As a result, an exhaustive search for *lagPatterns* is exponential. Here, we describe an efficient and scalable approach to prune the search space for both steps.

The rest of the chapter is organized as follows: Section 4.1 provides some preliminaries and detailed problem description. Section 4.2 describes the algorithms to discover motifs and $lagPatterns$ and the optimization strategies. Section 4.3 presents the experiment results. We summarize in Section 4.4.

4.1 Preliminaries

A **time series** $T = \{v[1], v[2], ..., v[n]\}$ with length $|T| = n$ is a sequence of regularly sampled real value observations where $v[i]$ is observation value at time i. We use $T[i]$ to denote a value at time i in time series T.

A **subsequence** of a time series, denoted as $T[i, j]$, is a subset of **contiguous** observations starting at time i and ending at time j, i.e., $T[i, j]$ = $\{T[i], T[i + 1], ..., T[j]\}$. It's length, denoted as $|T[i, j]|$, is equal to $j - i + 1$. In this chapter, we **normalize** subsequence. The normalized subsequence is given as $T[i, j]$ = $\{\frac{T[i]-\mu}{\sigma}, \frac{T[i+1]-\mu}{\sigma}, ..., \frac{T[j]-\mu}{\sigma}\}$, where μ and σ are the mean and standard deviation of $\{T[i], T[i + 1], ..., T[j]\}$ respectively. Here after, $T[i, j]$ refers to normalized subsequence.

A subsequence $T[i, j]$ is **similar** to another subsequence $T[p, q]$ if they have the same length and *dist*($T[i, j]$, $T[p, q]$) $\leq \delta$, where $dist(.)$ is Euclidian distance and δ is a user-defined distance threshold.

Normalizing time series (or subsequence of time series) helps to compare two time series (subsequence) having similar shape irrespective of their magnitude. For example, consider three time series shown in Figure 4.3 and its normalized version in Figure 4.4. Without normalization, $dist(T1, T2)$ = 2, $dist(T1, T3)$ = 3 and $dist(T2, T3)$ = 1. Note that, the shape of time series $T1$ and $T3$ is similar. Figure 4.3 shows the normalized version of these time series. Now, $dist(T1, T2)$ = 0.39, $dist(T1, T3)$ = 0 and $dist(T2, T3)$ = 0.39. Other benefits of normalization is explained in [47].

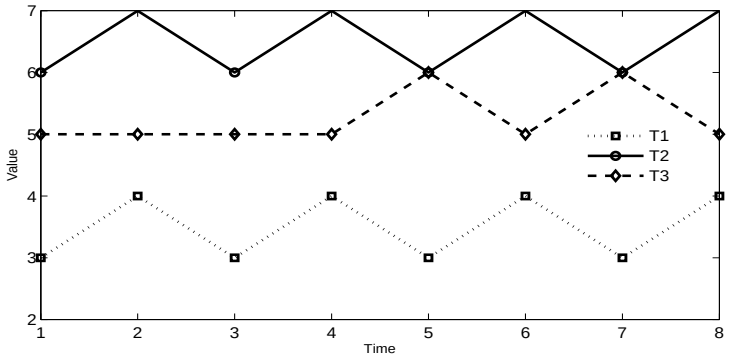


Figure 4.3. Time series.

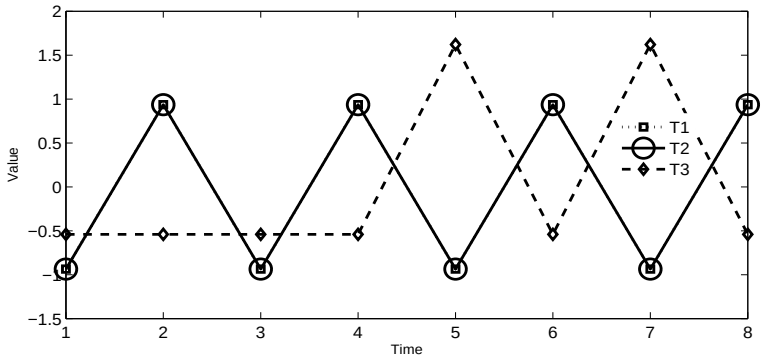


Figure 4.4. Normalized Time series.

Time Series	**Motifs m (correlation coefficient** $coef$ = 0.95)
T_1	m_{11} = {$\underline{T_1[14,17]}$, $T_1[1,4]$, $T_1[6,9]$, $T_1[22,25]$}
	m_{12} = {$\underline{T_1[22,25]}$, $T_1[3,6]$, $T_1[14,17]$}
	m_{13} = {$\underline{T_1[12,14]}$, $T_1[1,3]$, $T_1[22,24]$}
	m_{14} = {$\underline{T_1[6,9]}$, $T_1[14,17]$, $T_1[21,24]$}
T_2	m_{21} = {$\underline{T_2[15,17]}$, $T_2[2,4]$, $T_2[7,9]$, $T_2[23,25]$}
	m_{22} = {$\underline{T_2[17,20]}$, $T_2[6,9]$}
T_3	m_{31} = {$\underline{T_3[19,22]}$, $T_3[6,9]$, $T_3[11,14]$}
	m_{32} = {$\underline{T_3[4,7]}$, $T_3[9,12]$, $T_3[17,20]$}
T_4	m_{41} = {$\underline{T_4[20,23]}$, $T_4[7,10]$, $T_4[12,15]$}
T_5	m_{51} = {$\underline{T_5[20,23]}$, $T_5[3,6]$, $T_5[7,10]$, $T_5[14,17]$}

Table 4.1. Running example

Given a time series T, a **time series motif** $m_{T[i,j]}$, having $T[i,j]$ as anchor subsequence, is the set of non-overlapping subsequences[2] from T that are **similar** to anchor subsequence $T[i,j]$. For simplicity, we will use m in place of $m_{T[i,j]}$ when $T[i,j]$ is obvious. The size of motif m, denoted as $|m|$, is the number of subsequences in m.

The **support of time series motif** m with anchor subsequence $T[i,j]$, denoted as $mSup(m)$, is defined as

$$mSup(m) = \frac{|T[i,j]| * |m|}{|T|} \tag{4.1.1}$$

[2]We can use the optimal greedy-activity-selector solution in [26] to discover the maximum set of non-overlapping subsequences.

For example, Table 4.1 shows a subset of motifs for five time series of length 25. The anchor subsequence in each motif is underlined. The support of m_{11} is given by $mSup(m_{11})$ = $\frac{4*4}{25}$ = 0.64.

Given N time series $T_1, T_2, \cdots, T_N$, let M_i be the set of motifs from time series T_i. A ***lagPattern*** of length k is a pattern template consisting of k motifs from different time series and their lags. Formally,

$$p = (\{m_{y_1}, m_{y_2}, \cdots, m_{y_k}\}, \{l_{y_1}, l_{y_2}, \cdots, l_{y_k}\}), m_{y_i} \in M_{y_i}$$

$y_i \neq y_j$ for $i \neq j$ and m_{y_i} lags m_{y_1} by l_{y_i}, $y_i, y_j \in [1, N]$ and $i, j \in [1, k]$.

For example, $p1$ = ($\{m_{11}, m_{21}, m_{41}\}$, {0,1,6}) is a *lagPattern* of length 3, as motifs m_{11}, m_{21} and m_{41} are from time series T_1, T_2 and T_4 respectively. But, pattern $p2$ = ($\{m_{11}, m_{12}\}$, {0,8}) is not a *lagPattern* as both motifs are from the same time series T_1. Note that, the lag between two motifs in $lagPattern$ is a lag between start time of their respective anchor subsequences. For example, lag between motif m_{11} and m_{21} in $p1$ is a lag between $T_1[14, 17]$ and $T_2[15, 17]$. For this case it is 1.

A *lagPattern* $p1$ is a **subpattern** of another *lagPattern* $p2$ if all motifs in $p1$ also occurs in $p2$ with the same invariant ordering. For example, $p1$ = ($\{m_{11}, m_{41}\}$, {0,6}) is a subpattern of $p2$ = ($\{m_{11}, m_{21}, m_{41}\}$,{0,1,6}).

Now, we define two interesting measures for *lagPattern*. The first measure quantifies the frequency of *lagPatterns* and second measure suggest the association among motifs from *lagPattern*.

The **support of a *lagPattern*** p = ($\{m_1, m_2, \cdots, m_k\}$, $\{l_1,l_2,\cdots,l_k\}$), denoted as $pSup(p)$, is the size of the set $\{(s_1, s_2, \cdots, s_k) \mid (s_1 \in m_1) \wedge (s_2 \in m_2) \wedge \cdots \wedge (s_k \in m_k) \wedge (s_y$ lags s_1 by $l_y, 1 \leq y \leq k)\}$.

For example, consider p = ($\{m_{11}$, $m_{21}\}$,{0,1}). We observe that $T_2[7, 9] \in m_{21}$ lags $T_1[6, 9] \in m_{11}$ by 1. Similarly, $T_2[23, 25] \in m_{21}$ lags $T_1[22, 25] \in m_{11}$ by 1, $T_2[2, 4] \in m_{21}$ lags $T_1[1, 4] \in m_{11}$ by 1 and $T_2[15, 17] \in m_{21}$ lags $T_1[14, 17] \in m_{11}$ by 1. Hence, they support the *lagPattern* p. Thus, the support set is {($T_1[1, 4]$,$T_2[2, 4]$), ($T_1[6, 9]$,$T_2[7, 9]$), ($T_1[14, 17]$,$T_2[15, 17]$), ($T_1[22, 25]$,$T_2[23, 25]$)}. In this case, the support of p, $pSup(p)$, is 4. The support of $lagPattern$ captures the number of repetitions. In this chapter, we require that the pattern should be repeated at least more than one time.

Given a *lagPattern* p, the **participation ratio** of p is defined as

$$pRatio(p) = \frac{pSup(p)}{max_{m \in p}\{|m|\}} \tag{4.1.2}$$

For example, the $pRatio$ of p = ($\{m_{11}$, $m_{21}\}$,{0,1}) = $\frac{4}{max\{4,4\}}$ = 1. The $pRatio$ is a variant of the well-known All_confidence measure [44] in association-based correlation analysis. The $pRatio$ measure is anti-monotonic. This property allows us to prune away a large part of the search space. $pRatio(p)$ measure the association among the occurrence of motifs in the time series.

Theorem 4.1.1 *The participation ratio measure of a* $lagPattern$ *is anti-monotonic, that is, if a* $lagPattern$ p *satisfy* $pRatio(p) \geq min\_ratio$*, then any subpattern* p' *of* p *also satisfies* $pRatio(p') \geq min\_ratio$.

Proof Let a length k $lagPattern$ p = ($\{m_1, m_2, \cdots, m_k\}$, $\{l_1,l_2,\cdots,l_k\}$). We have

$$pRatio(p) = \frac{pSup(p)}{max_{m \in p}(|m|)}$$

Assume $lagPattern\ p'$ is a subpattern of $lagPattern\ p$. It is obvious that $pSup(p') \geq pSup(p)$. Also, $max_{m' \in p'}\ (|m'|) \leq max_{m \in p}(|m|)$. Hence, $pRatio(p') \geq pRatio(p)$.□

This implies we do not need to generate p if any subpattern p' of p does not satisfy the min_ratio constraint.

Given min_sup and min_ratio, a ***lagPattern*** p is **valid** if following all conditions are satisfied:

- $pRatio(p) \geq min\_ratio$, and
- $pSup(p) > 1$, and
- for all motifs m, $m \in p$, $mSup(m) \geq min\_sup$.

Given minimum support threshold min_sup, minimum participation ratio min_ratio and N time series of length L, we want to mine all **valid *lagPatterns*** of length k, $2 \leq k \leq N$.

4.2 Discover Lag Patterns

The discovery of ***lagPatterns*** involves two main steps. We need to first identify all the motifs of various lengths in each time series, and then determine groups of motifs from different time series having invariant orderings. Algorithm 6 summarizes our overall approach to mine $lagPatterns$. We call Algorithm FindMotifs for each time series to find all of its motifs(Line 5). Note that, M_i denotes the set of motifs generated from time series T_i. Lines 7-9 remove motif m if it does not satisfy the minimum support. Otherwise, we align m to a reference time point and insert it into an inverted index(Lines 10-12). Next, we invoke Algorithm LPMiner to obtain the valid $lagPatterns$ (Line 15). We will discuss the details of each algorithm in the following subsections.

Algorithm 6 Discover $lagPatterns$

Input: N, L, min_sup, min_ratio, $coef$, $minLen$, $maxLen$
Output: LP = set of $lagPatterns$

```
1: LP = φ
2: invIndex = φ
3: M = φ; // sets of sets of motifs
4: for i = 1 to N do {// N = Number of time series}
5:    M_i = FindMotifs(T_i, coef, minLen, maxLen);
6:    for each motif m in M_i do
7:       if mSup(m) < min_sup then
8:          M_i = M_i - {m};
9:       else
10:         align m to a reference time point t_p;
11:         insert m into invIndex;
12:      end if
13:   end for
14: end for
15: LP = LPMiner(N, L, min_sup, min_ratio, M); // L = Length of time series
16: return LP
```

4.2.1 Find All Motifs in a Time Series T

To find all motifs from time series T, we consider each subsequence of length between $minLen$ and $maxLen$ from T as an anchor subsequence and discover it's similar subsequences from T and then form a motif. Recall, subsequence s_1 is similar to subsequence s_2 if $dist(s_1, s_2) \leq \delta$. Since we consider anchor subsequences of various lengths, this δ threshold should be length-invariant[3]. Here we utilize the results in [102] which states that the Euclidian distance δ between two normalized time series of length len depends on their correlation coefficient $coef$, that is, $\delta = \sqrt{2 * (len - 1) * (1 - coef)}$. With this equation, we are able to employ the Euclidean measure in the similarity computation by setting the appropriate δ for varying length, given a fixed value of $coef$.

[3]Using single value of δ for mining motifs of various length might miss longer length motifs. At the same time, it is not feasible for user to provide δ value for each length of motif.

In this section, we describe a method that uses order line concept [64] and subsequence matching property [53] to find all motifs of length between $minLen$ and $maxLen$ from T efficiently. Assume, we are discovering motifs of length len from T. Given a set DB of normalized subsequences of length len from time series T and a pivot subsequence $s_p \in DB$. We obtain an **order line** by sorting the subsequences in DB according to their distance similarity from s_p.

For example, Figure 4.5(a) shows the distribution of subsequences of length 2 in a two-dimensional space. Assuming that the subsequence 2 is pivot subsequence, Figure 4.5(b) shows the order line. The number above the order line shows the subsequence id while the number below gives it's euclidian distance from pivot subsequence 2. Once order line is prepared, we discover similar subsequences for each anchor subsequence(i.e., each subsequence on order line).

We traverse the order line (with pivot subsequence s_p) from left to right. Given a distance threshold δ, suppose s_i is the next subsequence on the order line to be processed. We determine the similar subsequences of s_i by checking all the subsequences that fall within δ distance from s_i on the order line. This is due to the reverse triangular inequality which states that $dist(s_i, s_j) \leq \delta$ if and only if $|dist(s_p, s_i) - dist(s_p, s_j)| \leq \delta$.

Consider Figure 4.5(b). Let the subsequence we encounter be s_1 whose distance from the pivot subsequence s_2 is 2.24. If δ = 2, then a subsequence s is similar to subsequence s_1 if $dist(s_2, s)$ falls within [2.24-δ, 2.24+δ], that is, [0.24, 4.24]. Hence, the set of candidate similar subsequences for s_1 is $c_{s_1} = \{s_1, s_5, s_8\}$. We compute the actual distances between s_1 and each subsequences in c_{s_1} to obtain the final set of subsequences that are similar to s_1(i.e., a motif having anchor subsequence s_1).

Similarly, the set of candidate similar subsequences for s_5, $c_{s_5} = \{s_5, s_1, s_8, s_4\}$. Note that, we do not need to compute the actual distance between s_5 and s_1 since $dist(s_5, s_1) = dist(s_1, s_5)$

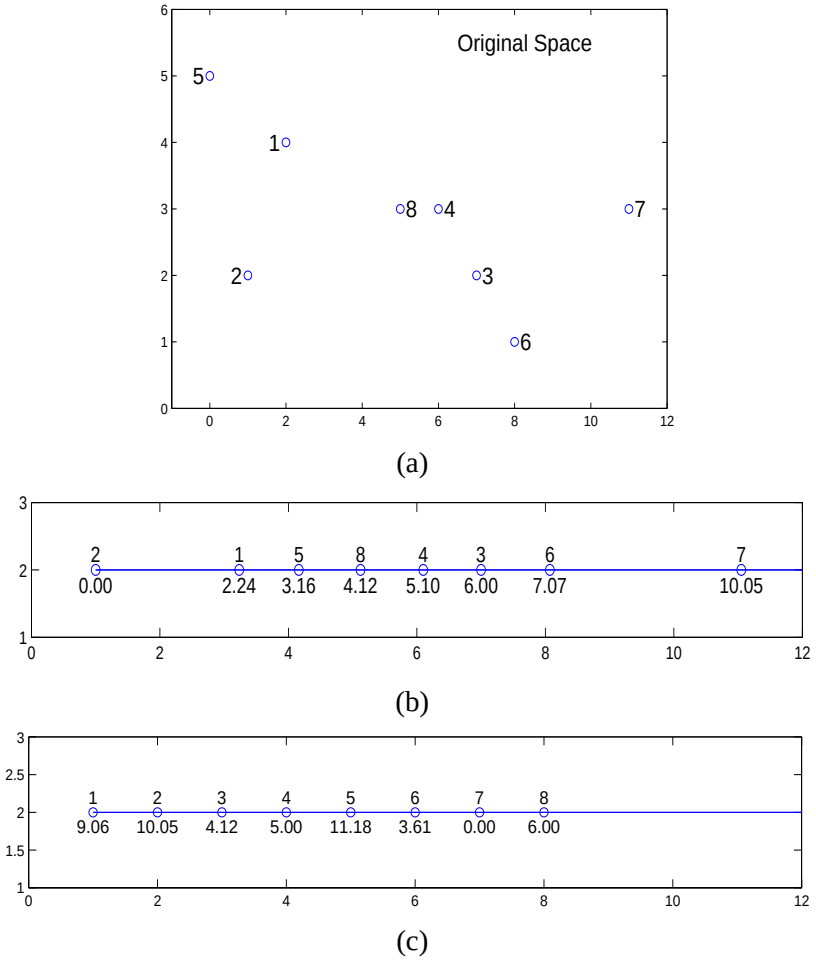


Figure 4.5. (a) Dataset of two-dimensional subsequences, (b) an ordering of subsequences with their distance value from subsequence 2 (c) distances of all subsequences from subsequence 7

and we have already obtained $dist(s_1, s_5)$ previously if s_1 and s_5 are similar. In other words, when traversing the order line from left to right, we need to perform the actual distance computations only for those candidates to its right. Clearly, we do not need to calculate the distance computations with all other subsequences from DB. Thus, order line concept helps to reduce the number of distance computations required during similarity search.

Another observation is that multiple order lines can prune more candidates. Suppose we have a second order line with pivot subsequence s_7 (see Figure 4.5(c)). Using the first order line(See

Figure 4.5(b)), we have the set of candidate similar subsequences for s_5, c_{s_5} = $\{s_5, s_1, s_8, s_4\}$. From the second order line, we observe that $dist(s_7, s_8) = 6$ and $dist(s_7, s_5) = 11.18$. Hence, $dist(s_8, s_5) \geq 5.18$ which is more than δ. The same process is repeated for subsequence s_4. Thus, applying triangular inequality, we eliminate s_8 and s_4 from c_{s_5} without performing any distance computation. In summary, the first order line is used to obtain initial candidate set of similar subsequences for any subsequence while the remaining order lines are used for further pruning. Algorithm 7 describes the detail of mining all motifs using the concept of order lines only.

Algorithm 7 OrderLine: Find All Motifs

Input: T, $coef$, $minLen$, $maxLen$, $numOrderLine$
Output: M = set of Motifs in T

1: $M = \phi$
2: **for** len = $minLen$; $len \leq maxLen$; len++ **do**
3: $\delta = \sqrt{2 * (len - 1) * (1 - coef)}$ // distance threshold for length len
4: DB = {normalized subsequences of length len from T}
5: Prepare $numOrderLine$ orderlines O
6: Let I denotes the first order line in O
7: **for all** subsequence $s \in DB$ **do**
8: c_s = {similar subsequences of s from DB using order line I}
9: refine c_s using remaining order lines in O
10: $s\_set = \{s' \mid s' \in c_s \wedge dist(s, s') \leq \delta\}$
11: Store s_set in M
12: **end for**
13: **end for**
14: return M

The order line based algorithm (Algorithm 7) efficiently finds all similar subsequences for a fixed length subsequences. However, In order to find similar subsequences for subsequence of length between $minLen$ to $maxLen$, we need to iterate the algorithm ($maxLen$ - $minLen$ + 1) times and prepare order line for these many times(Line 2). Thus, we integrate the subsequence matching property[53] with order line concept to reduce the number of iterations by 50%. The idea goes as follow: the order line prepared to find similar subsequences of length len subsequence is

also used to find similar subsequences of length $len + 1$ subsequence. Let δ be a distance threshold for mining motif of length len and ϵ be a distance threshold for mining motifs of length $len + 1$. The subsequence matching property states that,

$$dist(T[i, j + 1], T[i_1, j_1 + 1]) \leq \epsilon \Rightarrow dist(T[i, j], T[i_1, j_1]) \leq \epsilon'$$

where,

ϵ' = $\sqrt{2\omega - 2\sqrt{\omega^2 - \omega.\epsilon^2.\frac{\sigma^2(T[\text{i,j+1}])}{\sigma^2(T[\text{i,j}])}}}$,

ω = $|T[i, j]|$,

$\sigma^2(T[\text{i,j+1}])$ = standard deviation of un-normalized subsequence $T[i, j + 1]$,

$\sigma^2(T[\text{i,j}])$ = standard deviation of un-normalized subsequence $T[i, j]$.

This property is based on the observation that the occurrences of subsequences similar to $T[i, j + 1]$ coincides with the occurrences of subsequences similar to $T[i, j]$ most of the time. Hence, we can discover the candidate set of subsequences similar to subsequence $T[i, j + 1]$ while discovering set of subsequences similar to $T[i, j]$ by setting the appropriate distance threshold given by maximum$\{\delta, \epsilon'\}$. The new distance threshold, maximum$\{\delta, \epsilon'\}$, ensures that we do not miss any similar subsequence of $T[i, j]$ and $T[i, j + 1]$. In most cases, $\delta \leq \epsilon'$. With this, we present an exact algorithm **FindMotifs**(See Algorithm 8).

FindMotifs finds similar subsequences of subsequence $T[i, j]$ in a time series T. At each iteration, we set δ and prepare a database DB(Lines 3-4). Line 5 prepares order lines for subsequences of length len. Next, it invokes *GenerateMotif* to obtain all matches of every anchor subsequences of length len as well as the candidate sets for anchor subsequences of length $len + 1$. Line 10 prepares a database of subsequences of length $len + 1$. Finally, we call *RefineMotif* to eliminate

Algorithm 8 FindMotifs

Input: T, $coef$, $minLen$, $maxLen$, $numOrderLine$

Output: M = set of motifs in T

Set $M = \phi$ and $len = minLen$
while $len \leq maxLen$ **do**
  $\delta = \sqrt{2 * (len - 1) * (1 - coef)}$
  $DB \leftarrow$ {normalized subsequences of length len from T}
  Prepare $numOrderLine$ order lines O
  Let I denotes the first order line in O
  [M_{len}, C] = **GenerateMotif** (DB, I, O, len, δ)
  Set $M = M \cup M_{len}$ **and** $len = len + 1$
  $\delta = \sqrt{2 * (len - 1) * (1 - coef)}$
  $DB \leftarrow$ {normalized subsequences of length len from T}
  [M_{len}] = **RefineMotif** (DB, I, O, C, δ)
  Set $M = M \cup M_{len}$ **and** $len = len + 1$
end while
return M

Procedure GenerateMotif(DB, I, O, len, δ)

Let M be the set of motifs m_s for all $s \in DB$
Let C be the set of candidate subsequences for all $s \in DB$
Set $m = \phi$ and $c = \phi$ for all $m \in M$ and $c \in C$
for $j = 1$ to $|I|$ **do**
  select $s_j \in DB$ as an anchor subsequence
  Determine ϵ' using $len + 1$ and s_j
  $new\delta$ = max{ϵ', δ}
  $canSet$ = {candidate similar subsequences of s_j using I w.r.t. $new\delta$}
  $canSet$ = Refine $canSet$ using remaining orderlines
  for $s_k \in canSet$ **do**
    if $dist(s_k, s_j) \leq \delta$ **then**
      Add (s_k to m_{s_j}) and (s_j to m_{s_k})
    end if
    if $dist(s_k, s_j) \leq \epsilon'$ **then** Add (s_k to c_{s_j}) **end if**
  end for
end for
return M and C

Procedure RefineMotif(DB, I, C, δ)

Let M be the set of motifs m_s for all $s \in DB$
Set m to ϕ for $m \in M$
for $j = 1$ to $|I|$ **do**
  if $s_j \in DB$ **then**
    for each subsequence s in $c_{s_j} \in C$ **do**
      if $s \in DB$ and $dist(s, s_j) \leq \delta$ **then**
        Add (s to m_{s_j}) and (s_j to m_s)
      end if
    end for
  end if
end for
return M

the false matches found in the candidate sets obtained by *GenerateMotif* for length $len + 1$(Line 11).

The *GenerateMotif* procedure discovers similar subsequences of length len subsequence, that is, $T[i, i+len-1]$. At the same time, we also keep track of the candidate sets for subsequences of length $len + 1$, that is, $T[i, i + len]$. We use s_j to denote the j^{th} subsequence along the order line I. Next, we determine ϵ' and set the new distance threshold as $new\delta$(Lines 20-21). The $new\delta$ makes sure that we do not miss finding similar subsequence of $T[i, i+len-1]$ and $T[i, i+len]$. For each subsequence s_j on I, we obtain it's candidate set of subsequence similar to s_j using I(Line 22). Line 23 implements the triangular inequality based pruning and refine $canSet$. Finally, we compute the dist(s_j,s_k), $s_k \in canSet$. If $dist(s_k, s_j) \leq \delta$, we add s_k to the set of subsequence similar to s_j(i.e., m_{s_j}) and add s_j to m_{s_k} due to the **symmetry** property. In addition, if $dist(s_k, s_j) \leq \epsilon'$, then we add s_k to the candidate set c_{s_j}. Once all subsequences from I are processed, we return m_{s_j} and c_{s_j} discovered for all subsequences from DB.

The *RefineMotif* procedure finds all similar subsequences for length $len + 1$. Again, we traverse the order line I from left to right(Line 34). To find subsequences similar to s_j, we use the candidate set c_{s_j} obtained by *GenerateMotif*. Line 37 calculates $dist(s_j,s)$, s in c_{s_j}. If distance $dist(s_j,s) \leq \delta$, we add s to m_{s_j} and s_j to m_s.

4.2.2 Align Motifs

Having found the sets of motifs from each time series, the next step is to discover valid $lagPatterns$. A naive approach is to enumerate all possible combinations of motifs across multiple time series and calculate the support and participation ratio for each combination. Recall, enumerating all possible combination of motifs has an exponential time complexity. The anti-

$lagPattern$	Support Set
($\{m_{11}, m_{22}\}$,{0,3})	$\{(T_1[14, 17], T_2[17, 20])\}$
($\{m_{12}, m_{21}\}$,{0,-7})	$\{(T_1[22, 25], T_2[15, 17])\}$
($\{m_{12}, m_{22}\}$,{0,-3})	$\{(T_1[22, 25], T_2[17, 20])\}$

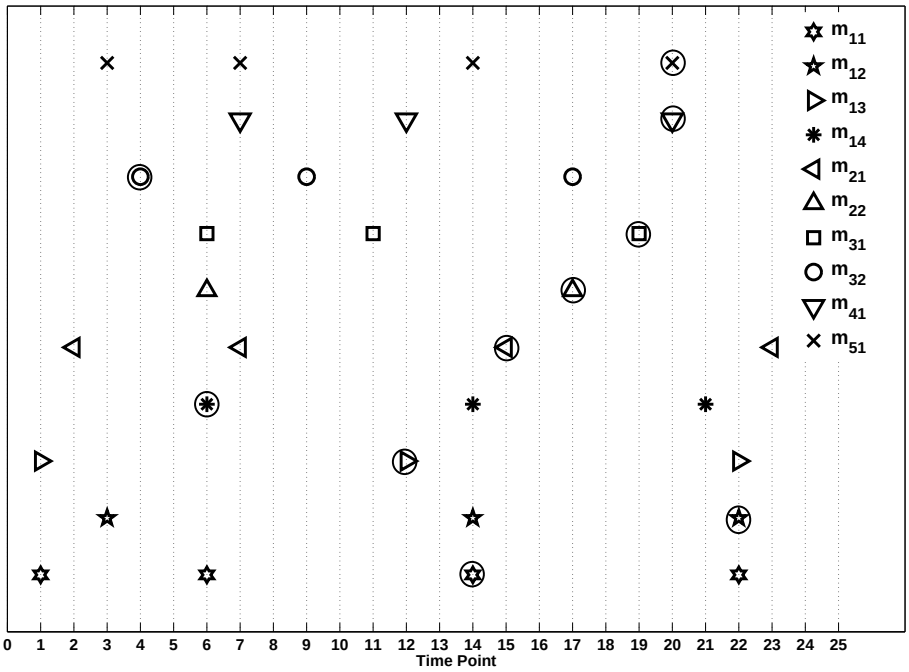
Table 4.2. Subset of lag patterns considered by naive enumeration

monotonic property of $pRatio$ that we have proved in Section 4.1 allows us to perform early elimination of $lagPatterns$ that cannot be valid. However, the naive approach still needs to form many $lagPatterns$ p of length 2 that has $pSup(p)$ = 1(i.e., no repetition) as shown in Table 4.2.

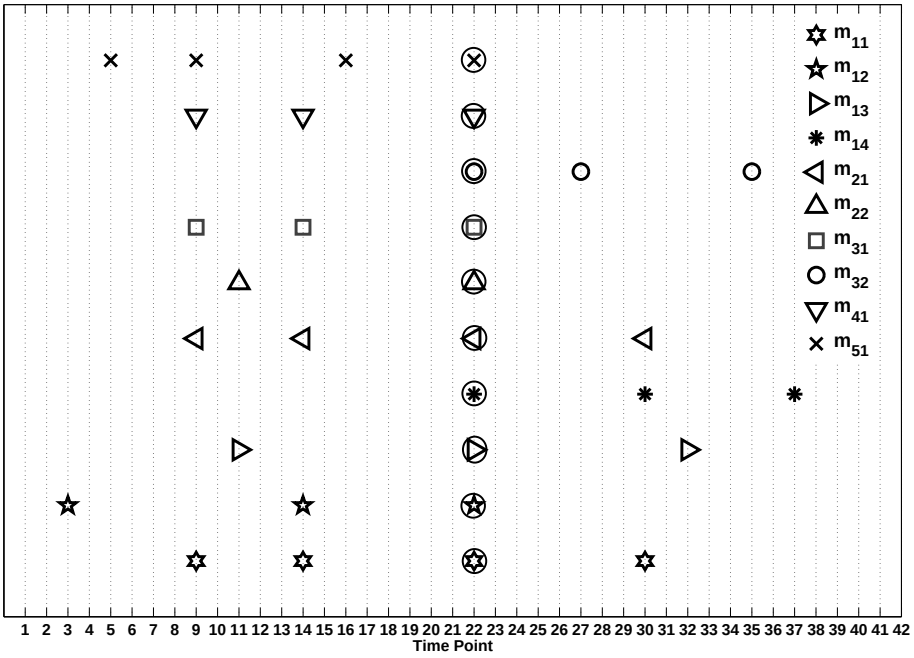
Further, computing $pRatio(p)$ of $lagPattern$ p is also a costly operation. In order to compute the $pRatio$ of a $lagPattern$ p = ($\{m_1, m_2, \cdots, m_k\}$, $\{l_1, l_2, \cdots, l_k\}$), we need to obtain the $pSup(p)$. The naive time complexity of computing $pSup(p)$ is O($|m_1| \times |m_2| \times ... \times |m_k|$) as explained in example I.

Example I: Consider finding support set of ($\{m_{11}, m_{22}\}$,{0,3}). Recall, m_{11} = $\{T_1[14, 17]$, $T_1[1, 4]$, $T_1[6, 9]$, $T_1[22, 25]\}$ and m_{22} = $\{T_2[17, 20]$, $T_2[6, 9]\}$. Thus, the set of possible combinations of subsequences from m_{11} and m_{22} are $\{(T_1[14, 17], T_2[17, 20])$, $(T_1[14, 17], T_2[6, 9])$, ...$\}$. However, only one combination from these possibilities, i.e. $\{(T_1[14, 17], T_2[17, 20])\}$, satisfies the lag relation between m_{11} and m_{22}.

To avoid enumerating all $lagPatterns$ and speed up the computation of $pSup(p)$, we align all motifs to some reference time point t_p. Aligning motif m means aligning it's anchor subsequence to t_p and shifting all it's similar subsequences accordingly. We set t_p to be the length of time series minus $minLen$(i.e., minimum length of motif). In our example, we choose t_p = 22 to align all motifs. Figures 4.6(a) and 4.6(b) show the motifs before and after alignment. The circled points denote the anchor subsequences. After alignment, each time point will show a list of motifs. We observe that the motifs, denoted by the symbols $\lhd$, $\square$ and ∇, occur together at time points 9,



(a) Before alignment



(b) After alignment

Figure 4.6. Motifs before and after alignment.

14, and 22. In other words, the $pSup(\{m_{21}, m_{31}, m_{41}\}, \{0, 4, 5\})$ is 3. The $pRatio$ of this pattern is $\frac{3}{max\{4,3,3\}}$ = 0.75. Now, we explain two benefits of alignment as follow:

- After alignment, the time complexity of computing $pSup(p)$ is O($|m_1| + |m_2| + ... + |m_k|$).

  – Consider finding support set of ($\{m_{11}, m_{22}\}$,{0,3}). After alignment, the start time of subsequence in m_{11} is {9, 14, 22, 30} and m_{22} is {11, 22}. Using hash join, we can get the intersection of these two sets in $O(|m_{11}| + |m_{22}|)$. For this case, it is {22}. From this result, we obtain the support set. The support set is $\{(T_1[14, 17], T_2[17, 20])\}$ as subsequence $T_1[14, 17]$ from m_{11} and subsequence $T_2[17, 20]$ from m_{22} are aligned at time point 22.

- The alignment of motifs provides us with information on which combinations of motifs are likely to form $lagPatterns$ p that can have $pSup(p)$ at least 2.

  – Consider motif m_{11}. After alignment, motif m_{11} has subsequence at time point 9, 14, 22, and 30. Careful observation of Figure 4.6(b) suggests that any motif that has subsequences at time point 9, 14 or 30 can form $lagPatterns$ with motif m_{11} and has $pSup(p)$ at least 2. In this case, the motifs are m_{21}, m_{31}, m_{41} and m_{51}. Note that, our approach does not form $lagPatterns$ with motif m_{22}.

To facilitate the support counting of *lagPatterns* and efficient discovery of *lagPatterns*, we construct an inverted index for the motifs occurring at each time point after the alignment. Fig. 4.7 shows the inverted index obtained from Fig. 4.6(b). Note that, at time point t_p(=22), all the motifs are present. In other words, all the $lagPatterns$ exist at time point t_p. We utilize this fact while

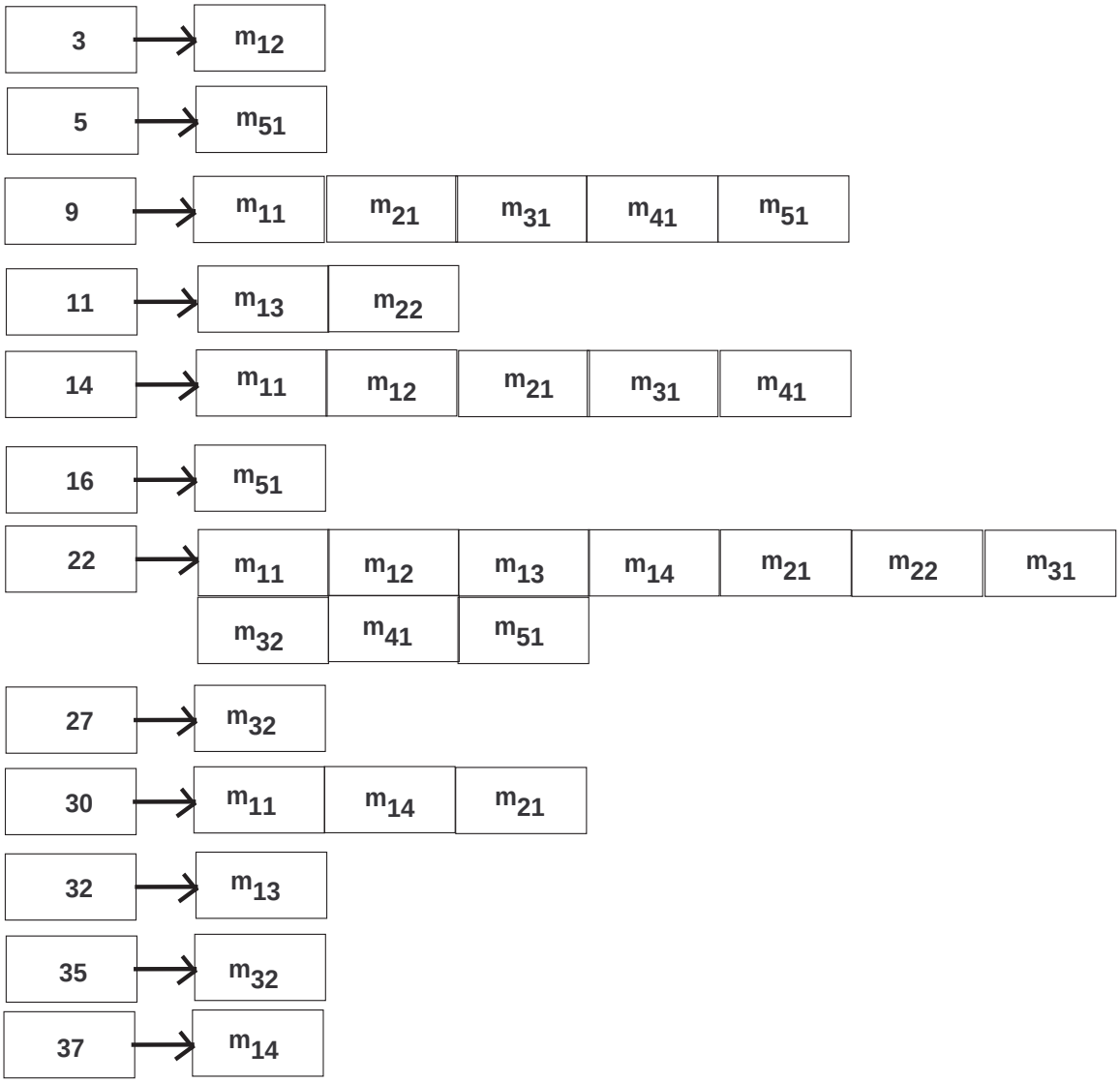


Figure 4.7. Inverted index for motifs in Fig. 4.6(b)

calculating the support of *lagPatterns*. Following the alignment, our method called $LPMiner$ utilizes the inverted index and search for valid *lagPatterns*.

4.2.3 Algorithm LPMiner

Method LPMiner processes each motif m and generates all length 2 *lagPatterns* from m as follows. For each motif m, we obtain the start times of its similar matches after alignment. These start times are used to probe the inverted index and to obtain all candidate motifs m'. Next, we form a $lagPattern$ between m and each candidate motif m', i.e., p = ($\{m, m'\}$,$\{l_1,l_2\}$). We also record the time points of the inverted index where the *lagPattern* p is generated. Those *lagPatterns* that satisfy the min_sup and min_ratio are valid and form the set of candidate patterns to generate longer *lagPatterns* (since *lagPatterns* are anti-monotonic).

Consider the motif m_{11}. After alignment, the start times of its matches are {9, 14, 22, 30} (see Fig. 4.6(b)). We probe the inverted index at time points 9, 14 and 30 respectively and obtain candidate motifs. In this case, the set of candidate motifs is $canSet$ = {m_{21}, m_{31}, m_{41}, m_{51}}[4]. Note that, there is no need to probe inverted index at the reference time point 22 since all motifs are aligned at this time point. In other word, any $lagPattern$ p is exists at this time point. The possible $lagPatterns$ are ($\{m_{11}, m_{21}\}$,{0,1}), ($\{m_{11}, m_{31}\}$,{0,5}), ($\{m_{11}, m_{41}\}$,{0,6}) and ($\{m_{11}, m_{51}\}$,{0,6}) as shown in Table 4.3. For each $lagPattern$, we have recorded the time points of the inverted index from where it is generated(See second column in Table 4.3). For example, the pattern p = ($\{m_{11}, m_{21}\}$,{0,1}) occurs at time points {9,14,22,30}. This implies $pSup(p)$ is 4. If min_ratio = 0.60, then $pRatio(p) = \frac{4}{max\{4,4\}} = 1 \geq min\_ratio$. Hence, it can be used to generate the longer patterns. Note that, all $lagPatterns$ except ($\{m_{11}$,$m_{51}\}$, {0,6}) satisfy min_ratio constraints and so they are valid $lagPatterns$.

[4]Without alignment method, all motifs from time series T_2, T_3, T_4 and T_5 are in $canSet$ for motif m_{11}. Thus, $|canSet|$ = 6 for naive method.

$lagPattern$	Time points	Support Set	$pSup$	$pRatio$
($\{m_{11}, m_{21}\}$,{0,1})	{9, 14, 22, 30}	$\{(T_1[1,4], T_2[2,4])$, $(T_1[6,9], T_2[7,9])$, $(T_1[14,17], T_2[15,17])$, $(T_1[22,25], T_2[23,25])\}$	4	1
($\{m_{11}, m_{31}\}$,{0,5})	{9, 14, 22}	$\{(T_1[1,4], T_3[6,9])$, $(T_1[6,9], T_3[11,14])$, $(T_1[14,17], T_3[19,22])\}$	3	0.75
($\{m_{11}, m_{41}\}$,{0,6})	{9, 14, 22}	$\{(T_1[1,4], T_4[7,10])$, $(T_1[6,9], T_4[12,15])$, $(T_1[14,17], T_4[20,23])\}$	3	0.75
($\{m_{11}, m_{51}\}$,{0,6})	{9, 22}	$\{(T_1[1,4], T_5[7,10])$, $(T_1[14,17], T_5[20,23])\}$	2	0.50

Table 4.3. Generated length 2 $lagPatterns$ using motif m_{11}

Let us consider the valid ***lagPattern*** p of length 2 = ($\{m_{11}$, $m_{21}\}$, {0,1}). Recall, this pattern is generated from time points 9, 14, 22 and 30. Hence, for this pattern, we again probe the inverted indexes at time points {9, 14, 30}(again no need to probe inverted index at time point 22) and obtain the candidate motif m' from time series T' with $T' > T_2$ for extension. In this case, the set of candidate motifs $canSet$ = $\{m_{31}, m_{41}\}$. Note that, motif m_{51} is not in $canSet$ as $lagPattern$ ($\{m_{11}$,$m_{51}\}$,{0,6}) does not satisfy the min_ratio. Hence, the possible length 3 ***lagPatterns*** are ($\{m_{11}$, m_{21}, $m_{31}\}$, {0,1,5}) and ($\{m_{11}$, m_{21} , $m_{41}\}$, {0,1,6}) both of which are generated from time points {9, 14, 22} and satisfy the min_ratio and hence valid. The process is repeated until no new valid pattern is obtained.

One key point is how support set of $lagPatterns$ is obtained using time points. Consider pattern p = ($\{m_{11}$, $m_{21}\}$,{0,1}) and it's time point {9, 14, 22, 30}. We know subsequence $T_1[1,4]$ from m_{11} and subsequence $T_2[2,4]$ from m_{22} are indexed at time point 9 after alignment. Thus, $(T_1[1,4], T_2[2,4])$ is in the support set of p. Next, subsequence $T_1[6,9]$ from m_{11} and subsequence $T_2[7,9]$ from m_{22} are indexed at time point 14. Hence, $(T_1[6,9], T_2[7,9])$ is also in the support set of p. Similarly, from time point 22 and 30, we obtain $(T_1[14,17], T_2[15,17])$ and $(T_1[22,25], T_2[23,25])$ respectively. Finally, the support set of p is $\{(T_1[1,4], T_2[2,4])$, $(T_1[6,9]$, $T_2[7,9])$, $(T_1[14,17], T_2[15,17])$, $(T_1[22,25], T_2[23,25])\}$.

Algorithm 9 *LPMiner*

Input: N, L, min_sup, min_ratio, M
Output: LP = set of $lagPattern$ = ϕ

1: **for** i = 1 to $N-1$ **do**
2: $motifSet$ = {motifs from M_i}
3: $extSet$ = {time series from T_{i+1} to T_N}
4: **for** each motif m in $motifSet$ **do**
5: Mine({m}, $extSet$)
6: **end for**
7: **end for**
8: return LP;

Procedure Mine(p, $extSet$)
9: $probeSet$ = {starting time points of p after alignment}
10: $canSet$ = ϕ
11: **for** each time point t in $probeSet$ **do**
12: **for** each m' in $invIndex[t]$ **do**
13: $canSet$ = $canSet \cup \{m'$, time point $t\}$
14: **end for**
15: **end for**
16: $extPattern$ = ϕ, $newExtSet$ = ϕ
17: **for** each entry $m' \in canSet$ **do**
18: p' = form $lagPattern$ between p and m'
19: **if** $pRatio(p') \geq min\_ratio$ **then**
20: $LP = LP \cup p'$
21: $newExtSet$ = $newExtSet \cup$ time series of m'
22: $extPattern$ = $extPattern \cup p'$
23: **end if**
24: **end for**
25: **for** each $lagPattern$ $lp \in extPattern$ **do**
26: **Mine**(lp, $newExtSet$)
27: **end for**

Algorithm 9 shows the details of LPMiner. Line 2 obtains all the motifs from M_i. $extSet$ maintains the list of time series from which the candidate motifs are obtained for extension(Line 3). For each motif m, we call procedure **Mine** to discover $lagPatterns$. The **Mine** procedure recursively extends the given *lagPattern* p. Line 9 obtains the time points of p to probe the inverted index. Lines 11-15 obtain all candidate motifs in $canSet$ using inverted index. Lines 17-24 generate the candidate $lagPattern$ between pattern p and each motif in $canSet$. The patterns satisfying

min_ratio are stored in LP (Line 20) and $extPattern$ (Line 22). The $Mine$ procedure is called recursively for each generated pattern in $extPattern$ (Line 26).

Algorithm LPMiner utilizes the anti-monotone property and inverted index to speed up the generation of $lagPatterns$. We derive an upper bound estimate of the participation ratio to further improve efficiency of LPMiner by pruning infeasible candidate patterns early.

Optimization. This optimization uses $|m_{T[i,j]}|$ to estimate the maximum $pRatio$ of a $lagPattern$ p = ($\{m_1, m_2, ..., m_k\}$, $\{l_1,l_2,...,l_k\}$). Since $pSup(p)$ must be less than or equal to $min_{m \in p}\{|m|\}$, the maximum $pRatio(p) \leq \frac{min_{m \in p}\{|m|\}}{max_{m \in p}\{|m|\}}$.

Consider $lagPattern$ p = ($\{m_{11}, m_{31}\}$, $\{0,5\}$). We have $|m_{11}|$ = 4 and $|m_{31}|$ = 3. Suppose the min_ratio is 0.80. Then the $pRatio(p)$ is $\frac{min\{3,4\}}{max\{3,4\}}$ = 0.75 ($<$ 0.80). Thus, this candidate is infeasible and can be removed from consideration for generating candidate $lagPatterns$.

For simplicity, LPMiner looks for exact lag among motifs. However, we can introduce a slack variable to relax this requirement. For example, LPMiner accesses inverted index at time points 11 and 32 to obtain candidates for m_{13}. However, with a slack value of 2, we now obtain possible candidates by accessing inverted index at time points $\{9,10,11,12,13\}$ and $\{30,31,32,33,34\}$. In this case, the pattern ($\{m_{13}, m_{21}\}$, $\{0,3\}$) will be in the output (See Fig. 4.6(b)).

4.3 Experimental Evaluation

We implement all our algorithms in C (compiled with GCC -O2). Our hardware configuration consists of a 3.2 MHz processor with 3GB RAM running Windows. We use synthetic datasets to verify the scalability of the proposed approach and real world datasets to demonstrate

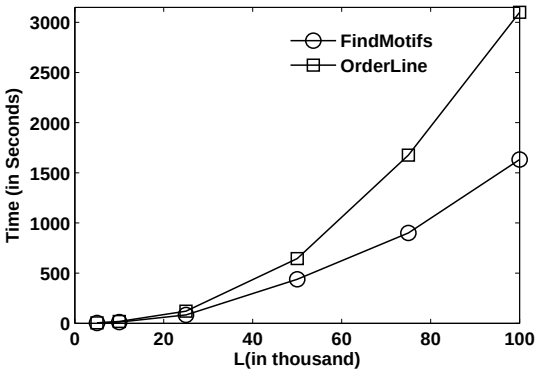
the usefulness of $lagPatterns$. A random walk generator [64, 26] is used to generate synthetic datasets D with N = 25 and L = 100000.

4.3.1 Efficiency Experiments

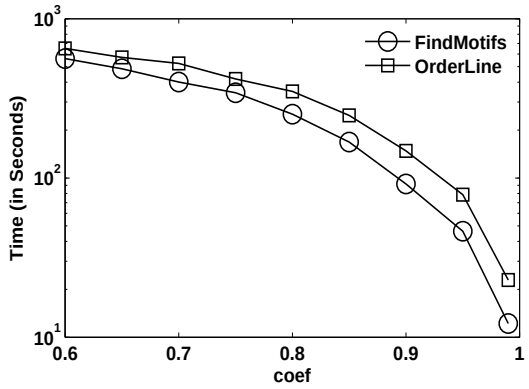
FindMotifs Algorithm. We select one time series from dataset D and apply FindMotifs algorithm to find all the motifs. We compare the performance of FindMotifs with algorithm OrderLine. The OrderLine algorithm uses only order line concept. The number of order lines is 5[64]. Fig. 4.8(a) shows the results of varying L from 5000 to 100000. We set $minLen$ = 99, $maxLen$ = 110 and $coef$ = 0.95. We observe that FindMotifs outperforms OrderLine, and the gap widens as the length of the time series increases.

Next, we set L = 20000 and vary the correlation coefficient $coef$ from 0.60 to 0.99. Fig. 4.8(b) shows the results in log scale. We observe that FindMotifs is much faster than OrderLine. In particular, when the correlation coefficient is greater than 0.9, FindMotifs is at least 50% faster than OrderLine. However, the gap narrows as $coef$ decreases. This is because FindMotifs estimates $new\delta(\geq \delta)$ in order to apply the subsequence matching property [53]. For low value of $coef$, $new\delta$ is much higher than δ resulting in a larger set of candidate subsequences for distance computation.

LPMiner Algorithm. Now, we report the results of our experiments on the datasets D. Unless otherwise stated, we set coef = 0.95, min_sup = 0.05, min_ratio = 0.80, N = 10, L = 10000, Min_Len = 99 and Max_Len = 110. Fig. 4.9 shows the results. Note that, running time does not include time required by FindMotifs algorithm. We observe that increasing L and N leads to an exponential increase in the runtime of LPMiner. This is expected since more *lagPatterns* will be generated with a large L and N. However, our optimization strategy is effective in cutting down the runtime. We also evaluate LPMiner by varying min_sup (see Fig. 4.9(d)) and min_ratio (see Fig.



(a) Effect of varying time series length



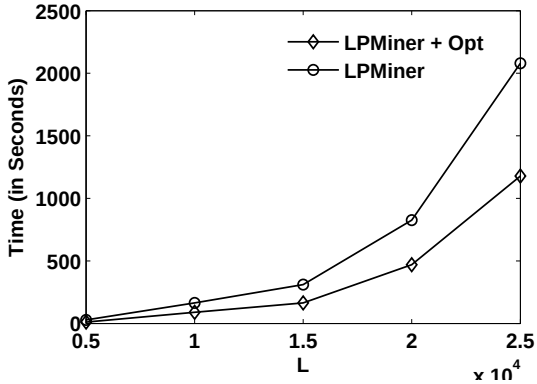
(b) Effect of varying coef

Figure 4.8. Runtime comparison between FindMotifs and OrderLine algorithms.

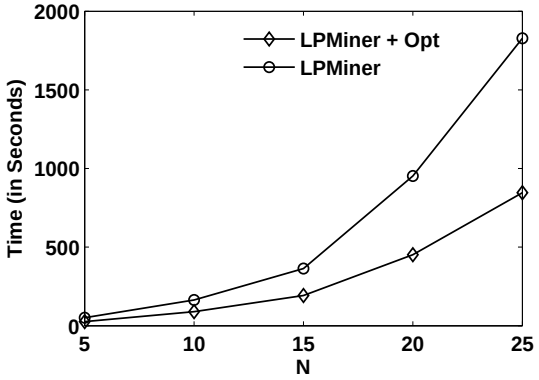
5.6(d)). Increasing min_sup reduces the number of subsequences and results in smaller inverted lists. Hence, the runtime decreases. Increasing min_ratio reduces the total number of possible valid *lagPatterns*, hence the runtime also decreases. Also, LPMiner takes less than one second to build an inverted index in all experiments. We also observed similar trends of LPMiner algorithm on real stock dataset, hepatitis dataset and stulong dataset.

4.3.2 Effectiveness Experiments

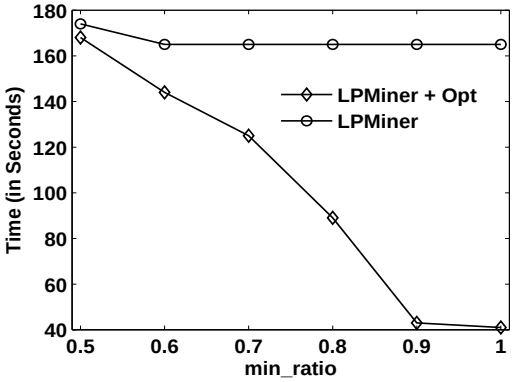
In this section, we mine $lagPatterns$ from real dataset from finance data and medical data and discuss usability of the discovered patterns.



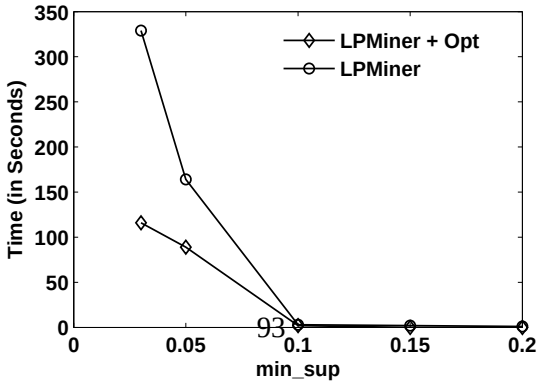
(a) Effect of varying L



(b) Effect of varying N



(c) Effect of varying min_ratio



(d) Effect of varying min_sup

Figure 4.9. Evaluation of LPMiner on dataset D.

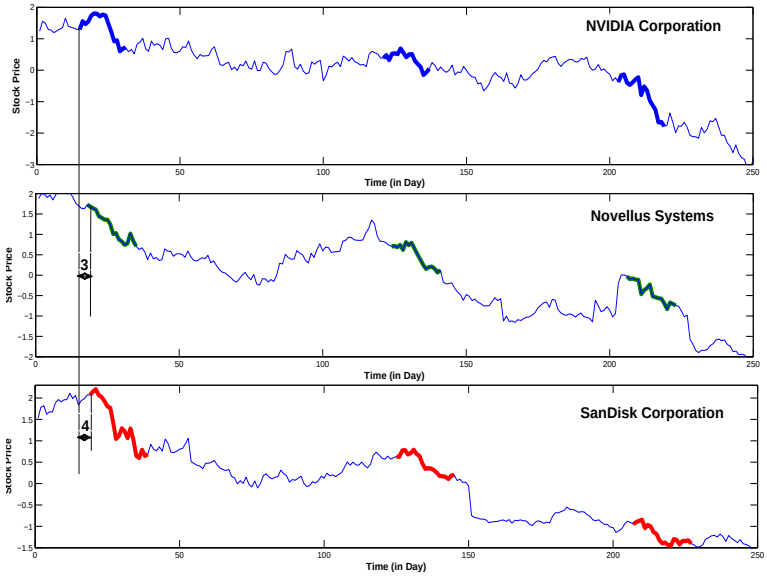
Finance dataset

We use S&P100 stock dataset[5](N=100, L=250) to find interesting localized associations among stock movements. Fig. 4.10(a) and Fig. 4.2 show examples of the discovered patterns. We observe that there is cooperative behavior among Nvidia, Novellus and SanDisk stocks. All these stocks are from semiconductor industry and none of them are competitor of each other. We use Yahoo Finance to verify competitor/co-operative behavior. To obtain these results, we set $coef$ = 0.90, min_sup = 0.10, min_ratio = 0.75, Min_Len = 6 and Max_Len = 21.

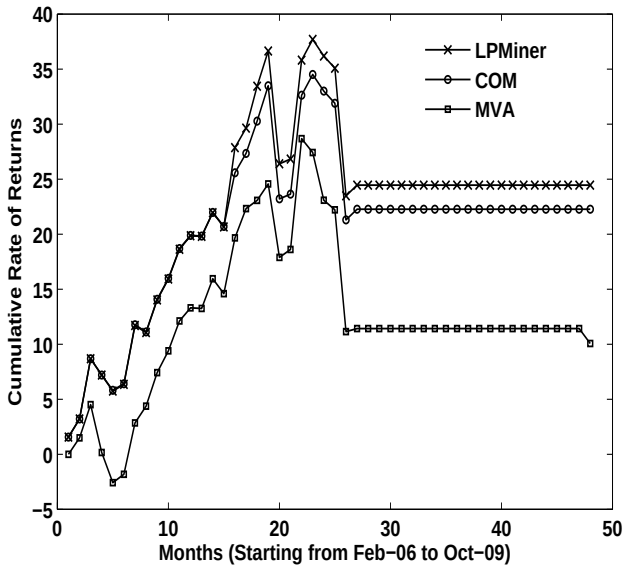
To further validate the effectiveness and utility of the discovered patterns, we construct a portfolio of equities selected from Morgan Stanley Capital International G7 (MSCI-G7) Index[6]. We use the equity indices of seven countries (Canada, France, Germany, Japan, Singapore, UK and USA) recorded daily over a 5 year period from March 2005 to October 2009(N=7, L=1260). The objective of a portfolio construction is to achieve a higher rate of return over a period of time (cumulative rate of return). Existing methods such Mean Variance Analysis(MVA) determine the investment weight for each equity indices from historical data.

Recently, an alternative method that updates the investment weights based on analyzing the co-movements of equities (COM) has been reported [92]. In order to leverage the $lagpatterns$, we first use the co-movement model to set the initial weights and subsequently utilize our *lagPatterns* to update the investment weights as described in [92]. Our $lagPatterns$ are obtained using LPMiner with $coef$ = 0.95, min_sup = 0.10, min_ratio = 0.80, $minLen$ = 3, $maxLen$ = 10, N = 7 and L = 240(one year window).

[5]http://biz.swcp.com/stocks/
[6]www.mscibarra.com



(a) Lag based motif association among Nvidia, Novellus and SanDisk stocks.



(b) Cumulative monthly rate of returns on MSCI-G7 Index.

Figure 4.10. Usability of $lagPatterns$ discovered from real world dataset.

We construct the portfolio for each month (March 2006 to October 2009) based on the data from the previous 12 months. We consider four week as one month. Fig. 4.10(b) presents the cumulative monthly rate of returns for MVA, COM and LPMiner. We observe that the cumulative rate of returns (over a period of 3 years) for LPMiner, COM and MVA is **26.64%**, 22.26% and 11.41% respectively. It is also important to note that this trend is observed across the board for most time points. The more than two-fold increase of LPMiner over MVA highlights the utility of our approach.

Significance of $lagPatterns$. Now, we verify the significance of $lagPatterns$ by shuffling the time series data using Fisher-Yates shuffle method [26]. The $lagPatterns$ are mined from the original dataset and shuffled dataset for the same set of parameters (See Table 4.4). We observe that, introducing randomness in the data significantly reduce the number of motifs and or $lagPatterns$. This shows that the discovered motifs and *lagPattern* are not due to random chance, but that they are meaningful patterns from the original time series, as we have significantly fewer patterns in the shuffled data. Similar observation is also found for the other parameters and datasets.

Dataset	**# Motifs**		# $lagPatterns$	
	Original Data	**Shuffled Data**	**Original Data**	**Shuffled Data**
S&P100 stock	110862	9166	2145943	1321
MSCI-G7 index	3535	2100	22	0
Hepatitis data	4010	39	353	0
Stulong data	5938	102	430	2

Table 4.4. The number of Motifs and $lagPatterns$.

Hepatitis dataset

In the second set of experiments, we utilize the time series data from Hepatitis dataset. This dataset has 490 patients. For each patient, we have 8 time series, one time series for one

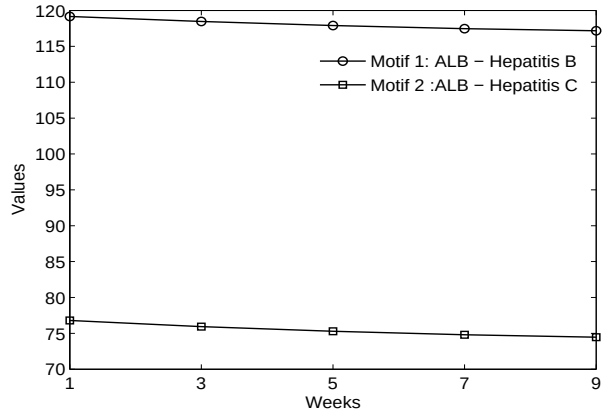
clinical attribute. For such dataset, we define motif as a subsequence that is repeated in the time series of the same attribute in many patients. Also the lag pattern is defined as an association among motifs derived from different clinical attributes. To use algorithm FindMotifs and LPMiner on such datasets, we fuse time series of same attribute from different patients. Thus, the modified dataset contains 10 time series, one time series for one clinical attribute.

Figure 4.11(a) presents an example of two time series motifs obtained from time series of attribute ALB. Both motifs represent behavior of result of ALB test over two months. The first motif, denoted as motif 1, appears in 28 patients, out of them 21 patients have Hepatitis B. Similarly, second motif, denoted as motif 2, appears in 18 patients, out of them 12 patients have Hepatitis C. From the first motif, we can derive a rule: patient having ALB value around 120 for 5 weeks will have higher chance of developing Hepatitis B.

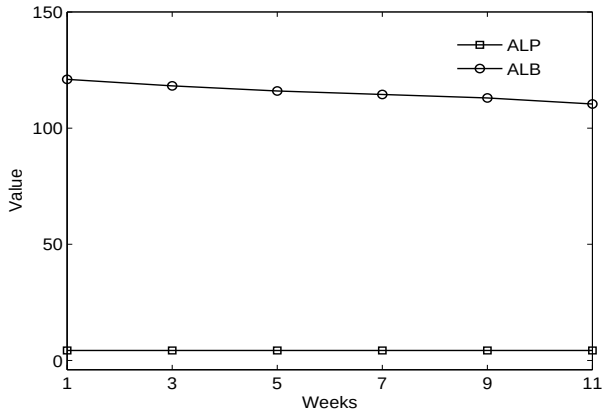
Next, Figure 4.11(b) presents an example of lag pattern. This lag pattern appears in 15 patients having Hepatitis B. It is not appeared in any patients from hepatitis C. Clearly, such information might be useful for time series classification. We have set $coef$ = 0.95, min_sup = 0.10, min_ratio = 0.65, Min_Len = 4 and Max_Len = 8 to achieve the above results. We have also observed that, shuffling the data does not generate motifs and lag patterns as shown in Table 4.4.

Stulong dataset

In the third set of experiments, we utilize the time series data from Stulong dataset. This dataset has 860 patients. For each patient, we have 10 time series, one time series for one clinical attribute. The definition of motif and lag pattern is similar as explained for Hepatitis dataset. To use algorithm FindMotifs and LPMiner on such datasets, we fuse time series of same attribute from



(a) time series motif

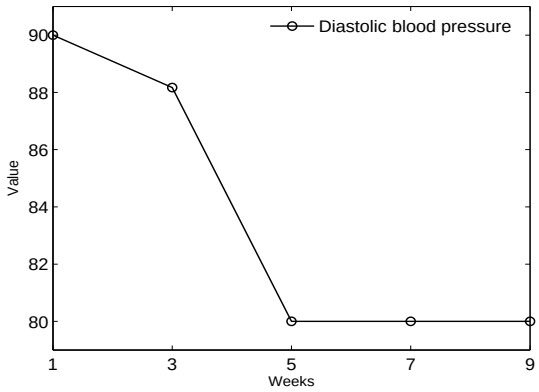


(b) lag patterns

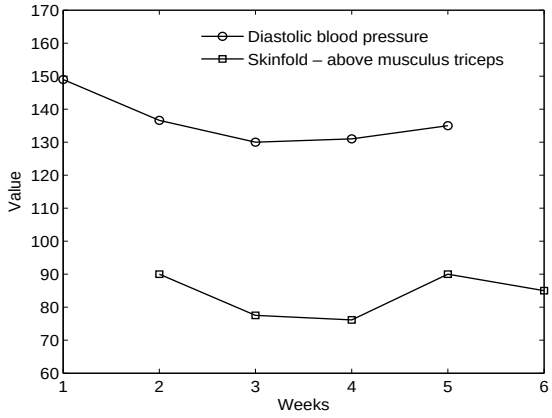
Figure 4.11. Example of motifs and lag patterns obtained from Hepatitis Dataset

different patients. Thus, the modified dataset contains 10 time series, one time series for one clinical attribute.

Figure 4.12(a) presents an example of a time series motif obtained from attribute diastolic blood pressure. This motif represents behavior of diastolic blood pressure test over two months. This motif appears in 32 patients, out of them 22 patients have no cardiovascular disease. From these motif, we can derive the following rule, a patient having diastolic blood pressure similar to Figure 4.12(a) will have less chance of developing cardiovascular disease. Next, Figure 4.12(b) presents an example of lag patterns. We set $coef$ = 0.95, min_sup = 0.10, min_ratio = 0.65, Min_Len = 4



(a) time series motif



(b) lag patterns

Figure 4.12. Example of motifs and lag patterns obtained from Stulong Dataset

and Max_Len = 8 to achieve the above results. We have also observed that, shuffling the data does not generate motifs and lag patterns(See Table 4.4).

4.4 Summary

In this chapter, we have introduced a new class of patterns called *lagPatterns* in time series data. The key contributions of this work are summarized as follows:

1. We define a new class of patterns, called as $lagPatterns$, to capture the orderings among motifs derived from different time series and prove that $lagPatterns$ satisfy the anti-monotonic property. This property allows us to prune the search space in the generation of $lagPatterns$. We design an efficient algorithm called $LPMiner$ that first aligns the motifs and then build an inverted index to quickly find group of motifs with invariant orderings.

2. We extend the exact motifs discovery algorithm in [64] to discover motifs of all lengths. We take advantage of order line concept and subsequence matching property of normalized time series to reduce over 60% of the distance computations.

3. We evaluate the algorithms on both synthetic and real world datasets. Our experimental results show that the proposed approach is scalable. We show the usefulness of *lagPatterns* discovered from a stock dataset by constructing stock portfolio that leads to a two-fold increase in the cumulative rate of return on investment compared to the traditional mean variance analysis(MVA) portfolio selection strategy.

In future, we would like to mine $lagPatterns$ from streaming data. This will be useful for financial application such as stock portfolio or investment suggestion for pair trading.

Chapter 5

Mine Patterns across Different Kinds of Data

We have seen that existing frequent pattern mining algorithms are geared toward finding frequent patterns from categorical data, numerical data and sequence data. The previous two chapters present frequent pattern mining algorithms for interval data and time series data. However, many database applications in the clinical and bioinformatics domains involve records with multiple kinds of data. For example, a patient's record typically comprises of categorical data, numerical data, time sequence data, interval data and time series data(See Table 5.1). Knowing the relationships among patterns from these different kinds of data can aid in the understanding of a patient's health condition.

Consider the two patterns

$\{Male, Smoking\}$ and

$\{Headache\ Overlap[0, 0, 0, 1, 0]\ HighBloodPressure\}$.

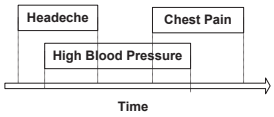
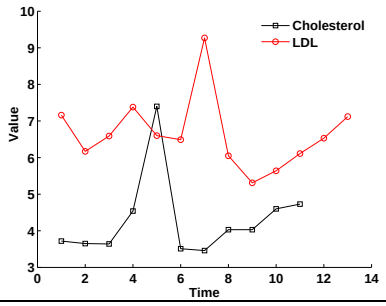
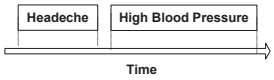
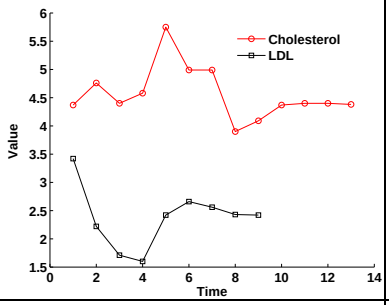
Id	Categorical Data	Numerical Data	Interval Data	Time Series Data	Class
1	Male, Smoking, Wine	Age = 21, DailyWineIntake = 2, AvgSysBldPre = 2			CVD = Yes
2	Male, NoSmoking, NoWine	Age = 30, DailyWineIntake = 0, AvgSysBldPre = 5			CVD = No
...	...	...	...	...	...

Table 5.1. Dataset with multiple kinds of data

The pattern $\{Male, Smoking\}$ is a frequently occurring itemset [8]. Well-known algorithms such as FPTree [39] can be utilized to find such frequently occurring itemsets. On the other hand, the pattern $\{Headache\ Overlap[0,0,0,1,0]\ HighBloodPressure\}$ is an interval-based temporal pattern and its discovery requires a totally different algorithm [75]. Separately, these patterns may not raise any alarm as there are many male smokers in the population who go about their daily lives normally. Similarly, many people suffer from headache with elevated blood pressure but they do not experience any serious consequences. However, the combination of these two patterns reveals a different picture. Studies have shown that a male smoker who experiences headache with elevated blood pressure is a likely candidate for cardiovascular diseases. We call this combination of patterns from different kinds of data as **heterogenous** patterns.

In order to mine heterogenous patterns, we must apply different algorithms for the different kinds of data. This is then followed by an exhaustive combination from each kind of data to form heterogenous patterns. However, an exhaustive combination is not a feasible solution. For example, a small dataset with 10 categorical attributes, 20 numerical attributes, 10 events, and 10 days of 10 time series data can result in the generation of 2^{10} frequent itemsets [96], 2^{20} frequent intervals, 10^{10} frequent temporal patterns, and 10^{10} time series motifs. Hence, the combination of these patterns is of the order $O(2^{58})$. Clearly, for practicality, we need an efficient algorithm to prune the search space.

We have seen that, early works on discovering heterogenous patterns are limited to mining patterns from at most two different kinds of data [79, 41]. The UniSeq algorithm [79] mines patterns from both categorical and sequence data while the MergeSD algorithm [41] is designed for both categorical and numerical data. These algorithms are based on exhaustive enumeration. Thus, they cannot be extended to discover patterns involving more than two kinds of data.

In this work, first we present a pattern mining algorithm, called **HTMiner**(**He**terogenous Pattern **Miner**), to mine all frequent heteorgenous patterns from dataset with multiple kinds of data. HTMiner is an integrated algorithm that systematically discovers frequent heterogenous patterns in a depth-first manner from a dataset consisting of categorical data, numerical data, interval data and time series data. Given a *minsup* threshold, HTMiner first discovers a set of frequent patterns from categorical data, numerical data, interval data and time series data. Next, HTMiner utilizes the computations performed in the previous stage to quickly prune off infeasible combinations for mining heteorgenous patterns. Our experimental results show that the proposed algorithm is very efficient.

In many real-world applications, the number of frequent patterns mined for various parametric settings is extremely large and only a subset of these patterns are useful. For example, frequent pattern based classifier uses only subset of frequent patterns for classification. Thus, we also present another mining strategy, called **HTClassifier**, to mine the essential set of discriminative heteorgenous patterns from dataset with multiple kinds of data for classification. HTClassifier is an iterative algorithm. In each iteration, HTClassifier discovers an essential heterogenous pattern for classification and performs instance elimination. This instance elimination step reduces the problem size progressively by removing training instances which are correctly covered by the discovered essential heterogenous pattern. Experiments on two real world datasets show that the classifier based on discovered patterns can significantly improve the classification accuracy.

To the best of our knowledge, this is the first work that integrates existing frequent pattern mining algorithms for single data kind to discover heterogenous patterns from datasets with multiple kinds of data. We demonstrate the effectiveness of such heterogenous patterns for classification by building two classifiers and comparing them with classifiers that are built using only patterns involving single kind of data. Experiment results on two real world datasets show that HTMiner is efficient and scalable in discovering heterogenous patterns. Further, the classifiers based on heterogenous patterns significantly outperform classifiers based on patterns involving at most two kinds of data.

The remaining of the chapter is organized as follow: In section 5.1, we describe the preliminary and the problem statement. HTMiner algorithm is explained in section 5.2. HTClassifier algorithm is explained in section 5.3. An extensive experimental result is reported in section 5.4. Section 5.5 summarizes the discussion.

5.1 Preliminaries

Let DB be a dataset with multiple kinds of data, namely, categorical, numerical, interval and time series. Each instance in DB has a class label. Table 5.2 shows an example of such a dataset where column 1 is the instance id, column 2 is the categorical data, column 3 is the numerical data, column 4 is the interval data, column 5 is the time series data, and finally column 6 shows the class labels.

DB can be projected into 4 datasets according to the kind of data as follows:

1. A categorical dataset, denoted as DB_C, contains instances of the form $< tid, C, class >$ where tid is the instance identifier, C is a set of items where each item is a categorical attribute-value pair, and $class$ is the class label of the instance.

2. A Numerical dataset, denoted as DB_N, contains instances of the form $< tid, N, class >$ where N is a set of numerical attribute-value pairs.

3. An Interval dataset, denoted as DB_I, is a set of instances of the form $< tid, E, class >$ where E is a list of events. Each event is a triplet $(\text{type, start, end})$. Events in E are sorted by their start times, end times and event types. Each pair of events $E_1, E_2 \in E$ has a temporal relationships [12] given by ($E_1.type\ R\ E_2.type$) where $R \in \{$Equal, Meet, Before, Start, Overlap, Contain, Finish_By$\}$.

4. A Time series dataset, denoted as DB_T, is a set of instances of the form $< tid, T, class >$ where T is a set of time series of length n. One time series represents one attribute. Each time series is of the form ($v[1]$, $v[2]$,..., $v[n]$) where $v[i]$ is the value of the time series at time point i. In this chapter, we use T_{XY} to refer a time series of attribute Y from instance X.

Instance identifier	Categorical data	Numerical data	Interval data	Time series data	Class
1	A,B,E,G	($attr_1$,0.4),($attr_2$,0.6)	(L,1,6),(K,4,8),(S,8,10),(T,11,12)	T_{11} = {5,6,7,10,14} T_{12} = {7,6,5,5,12}	1
2	A,B,D	($attr_2$,0.8),($attr_3$,0.2)	(L,1,5),(T,2,7),(K,3,9),(S,9,13)	T_{21} = {1,5,6,9,0} T_{22} = {10,7,6,4,0}	1
3	A,B,D	($attr_1$,0.38),($attr_3$,0.5)	(L,3,8),(T,3,9),(K,5,10),(S,10,12)	T_{31} = {4,5,7,0,4} T_{32} = {12,6,6,5,1}	1
4	B,D,E,G	($attr_2$,0.5),($attr_3$,0.9)	(K,2,4),(S,6,10)	T_{41} = {1,2,2,3,4} T_{42} = {0,5,0,1,0}	1
5	A,B,D	($attr_1$,0.37),($attr_3$,0.6)	(L,1,6),(K,3,8),(S,4,10)	T_{51} = {3,6,7,7,1} T_{52} = {0,8,0,6,0}	0
6	A,D,E	($attr_1$,0.27),($attr_2$,0.4)	(N,5,10)	T_{61} = {4,5,4,7,4} T_{62} = {0,4,0,3,0}	0

Table 5.2. Example: Dataset with multiple kinds of data

Different types of patterns can be discovered from each of the projected dataset. For the **categorical dataset** DB_C, we can find the set of frequent itemsets, where each itemset is a set of items that occurs together frequently. An instance of the dataset DB_C supports itemset I if I is a subset of the instance's item-set. The support of I is the ratio of the number of instances that are superset of I to the total number of instances in DB_C. For example, in Table 5.2, instance 1 supports the itemset {A, B}, but instance 4 does not. The support of itemset {A, B} is $\frac{4}{6}$ as instances 1, 2, 3 and 5 supports {A, B}. The length of itemset is the number of items it contains. For example, length of itemset {A, B} is 2.

For the **numerical dataset** DB_N, the patterns found are of the form np = {($attr_1, [l_1, u_1]$), ($attr_2, [l_2, u_2]$)$\cdots$} where l_i and u_i denote the lower and upper bound for the numerical attribute $attr_i$. We say that an instance I satisfies the pattern np if and only if for each attribute appearing in np, the corresponding value of that attribute in I lies in the specified range $[l, u]$ in np. For example, instance 1 in Table 5.2 satisfies np = {($attr_1, [0.39, 0.41]$),($attr_2, [0.58, 0.62]$)} as the value of $attr_1$($attr_2$) in instance 1 is 0.4(0.6) which is in [0.39,0.41]([0.58,0.62]). The support of np is the ratio of the number of instances that satisfies np to the total number of instances in DB_N.

The length of numerical pattern np is the number of attributes it contains. For example, length of $\{(attr_1, [0.39, 0.41])$,$(attr_2, [0.58, 0.62])\}$ is 2.

For the **interval dataset** DB_I, we can find sequence pattern, temporal pattern, annotated sequence pattern, and annotated temporal pattern. These patterns are defined as follows:

1. A temporal pattern ip has the form $(E_1.type \overset{R_1[c,f,m,o,s]}{\longrightarrow} E_2.type \overset{R_2[c,f,m,o,s]}{\longrightarrow} \cdots\ E_n.type)$ where $E_i.type$ is an event type and $R_i[c, f, m, o, s]$ is temporal relationships between event of type $E_{i+1}.type$ and all of it's preceding events in ip as explained in chapter 3.

2. A sequence pattern is the special case of the temporal pattern where each R_i in ip is limited to the "Before" relationship, denoted by ($E_1.type \rightarrow E_2.type \rightarrow ... \rightarrow E_{i+1}.type$).

3. An annotated temporal pattern is simply a temporal pattern with a time lag $t(\geq 0)$ is specified between adjacent pair of events in ip, denoted by $(E_1.type \overset{R_1[c,f,m,o,s](t_1)}{\longrightarrow} \cdots\ E_i.type \overset{R_i[c,f,m,o,s](t_i)}{\longrightarrow} E_{i+1}.type \cdots\ E_n.type)$.

4. An annotated sequence pattern is the special case of the annotated temporal pattern where each R_i in ip is limited to the "Before" relationship, denoted by $E_i.type \overset{t}{\rightarrow} E_{i+1}.type$.

A database instance from DB_I satisfies a temporal pattern ip if and only if for each pair of event types in ip, their respective start and end times in instance conforms to the corresponding temporal relationship in ip. Similarly, for an annotated temporal pattern with time lag t between a pair of events (E_i, E_{i+1}), we say that an instance satisfies this pattern if the instance satisfies the corresponding temporal pattern and the difference in the start times of these events (i.e., start time of E_{i+1} - start time of E_i) lies in the range $[t - \delta, t + \delta]$, where $\delta(\geq 0)$ is a threshold for the time lag. The length of ip is the number of events it contains.

Consider the annotated pattern ip = $\{L \overset{Overlap[0,0,0,1,0](2)}{\longrightarrow} K\}$where event of type L overlaps event of type K. Further, when event of type L occurs, event of type K will occur with a lag of 2 days (i.e., t=2). Suppose δ = 1, then instance 1 in Table 5.2 supports this ip, as event L at offset 1 overlaps event K at offset 2 and the difference in the start time of event L and event K in instance 1 is 3 which is within the range [1,3](i.e., 3 $\in$ [2-1,2+1]). However, instance 1 does not support $\{L \overset{Overlap(5)}{\longrightarrow} K\}$ as the difference in the start times of these events is not within the range [4,6] (i.e., 3 $\notin$ [5-1,5+1]). We compute the support of a temporal pattern as the ratio of the number of instances that satisfies the temporal pattern to the total number of instances in DB_I. The length of ip is 2.

For the **time series** dataset DB_T, a time series motif is the set of non-overlapping subsequences such that the subsequences in the set are of the same length, are from the same attribute and the distances between any pair of subsequences are less than a given threshold. For example, consider motif m_1 = $\{T_{11}[1,3], T_{31}[1,3], T_{61}[2,4]\}$ from attribute 1. This motif has subsequences from instance 1, 3 and 6. We use euclidian distance to measure the distance between two subsequences. The **support** of time series motif is the ratio of the number of instances that has subsequence in the set to the total number of instances in DB_T. For example, the support of m_1 is $\frac{3}{6}$. With these frequent motifs, we can discover motif sequences and annotated motif sequences from time series dataset DB_T. Motif sequences and annotated motif sequences(i.e., $lagPatterns$) have the same definition as sequence pattern and annotated sequence pattern where event type E is replaced by time series motif. The support of these patterns from time series data can be defined in a similar manner as the support of sequence patterns.

A **heterogenous** pattern, or **HT** pattern in short, is a quadruplet represented as $[cp, np, ip, tp]$, where cp, np, ip, tp are the patterns discovered from categorical, numerical, interval and time series data respectively.

For example, α = [{Smoking, Male}, $\emptyset$, $\{Smoking \rightarrow ChestPain\}$, $\emptyset$] is a HT pattern of size 2 since it has patterns only from the categorical and interval data. We use k-HT to refer a HT pattern of size k.

The support of a HT pattern α, denoted as ***sup(α)***, is the ratio of the number of instances in DB satisfying α to the total number of instances in DB. The confidence of α for a class label $class$, denoted by ***conf(α, $class$)***, is the ratio of the number of instances with class label $class$ satisfying α to the number of instances that satisfies α. For example, support of α = $[\{A, B\}, \phi, \{L \xrightarrow{Overlap[0,0,0,1,0]} K\}, \phi]$ is $\frac{4}{6}$, as instances 1, 2, 3 and 5 satisfy α. Also, *conf(α, 1)* = $\frac{3}{4}$.

A HT pattern α is ***frequent*** if sup(α) $\geq minsup$. A HT pattern α is ***discriminative*** if its confidence for one class is higher than max_conf and its confidences for the rest of the classes are lower than min_conf. A HT pattern α is an ***essential*** pattern for classification if it is both frequent and discriminative.

Given a minimum support threshold $minsup$ and dataset with multiple kinds of data DB, our purpose is to find the set of frequent heteorgenous patterns. Further, given a minimum support threshold $minsup$, maximum class confidence threshold max_conf, minimum class confidence threshold min_conf and dataset with multiple kinds of data DB, we discover a set of essential heterogenous patterns for classification.

5.2 Algorithm HTMiner

In this section, we present an algorithm, called HTMiner, to mine all frequent heterogenous patterns(see Algorithm 10). HTMiner first projects the dataset DB into the respective projected databases based on the kinds of data that exists in DB(Line 2). It then calls procedure MineSingle to generate a set of frequent patterns $patternSet$ along with their support values for each projected databases (Line 3). Details of MineSingle are given in Section 5.2.1. Line 7 invokes procedure MineMultiple to generate combination of patterns from different kinds of data. Details are given in Section 5.2.2. At last, the generated frequent patterns are outputted.

Algorithm 10 HTMiner

Input : database DB, minimum support threshold $minsup$
Output : set of frequent patterns $frePat$
Global variable : $frePat = \emptyset$, N = 4, $patternSet = \emptyset$

1: **for** p = 1 to N **do**
2: DB_p = projected database of DB for p^{th} kind of data
3: $patternSet_p$ = **MineSingle**(DB_p, $minsup$)
4: $frePat = frePat \cup patternSet_p$
5: **end for**
6: Remove infrequent attributes from DB
7: $resultSet$ = **MineMultiple**(DB, $minsup$)
8: return $\{frePat \cup resultSet\}$

5.2.1 Algorithm MineSingle

Algorithm MineSingle (see Algorithm 11) unifies the various frequent pattern mining algorithms for different kinds of data via a modified pseudo projection based pattern growth approach [77]. It grows the pattern in two dimensions: increase the length of pattern by 1, and increase the information content of pattern. In the first case, we call it an **extension** of the pattern. For example, itemset {A, B} is one possible extension of itemset {A}. Also, a temporal pattern {P

$\overset{Overlap[0,0,0,1,0]}{\longrightarrow} Q\}$ is one possible extension of temporal pattern $\{P\}$. In the second case, we call it an **annotation** of the pattern. For example, $\{P \overset{Overlap[0,0,0,1,0](7)}{\longrightarrow} Q\}$ is one possible annotation of $\{P \overset{Overlap[0,0,0,1,0]}{\longrightarrow} Q\}$. Only patterns from interval data and time series data require annotation.

Algorithm 11 MineSingle(DB_p, $minsup$)

Output : A frequent patterns $patSet$

1: $patSet = \emptyset$ // set of generated patterns
2: $freePatSet$ = {length 1 frequent patterns from DB_p}
3: **while** $freePatSet \neq \emptyset$ **do**
4: Select pattern α from $freePatSet$ and remove it from $freePatSet$
5: Add α to $patSet$
6: $extPatSet$ = {**extend** pattern α}
7: $freePatSet = freePatSet \cup extPatSet$
8: $annPatSet$ = {**annotate** pattern α}
9: $patSet = patSet \cup annPatSet$
10: **end while**
11: return $patSet$

Initially, Algorithm MineSingle obtains all the frequent patterns of length 1 from the given dataset with single kind of data DB_p (Line 2). Table 5.3 shows a subset of the length 1 patterns generated for each kind of data using dataset given in Table 5.2.

For **categorical data**, a length 1 pattern α is simply an item (i.e., attribute-value pair). We can determine its frequency by counting the number of instances in given database DB_C where this item occurs. At the same time, the instance id and the offset of the item α within the instance is kept in an index, called $index_\alpha$ for subsequent use(See Table 5.3(a)).

For **numerical data**, a length 1 pattern is of the form $(attr, [l, u])$. To discover the frequent range of an attribute $attr$, let V = [v_1, v_2, ..., v_k] be the possible values of $attr$ in DB_N. Given a similarity threshold δ, we restrict the numerical ranges as follows [$v_1-\delta$,$v_1+\delta$], [$v_2-\delta$,$v_2+\delta$], ..., [$v_k-\delta$,$v_k+\delta$]. For each numerical range, i.e., an interval item ($attr$,[$v_i-\delta$,$v_i+\delta$]), we determine

(a) Categorical dataset DB_C

A	
instance_id	offset_list
1	1
2	1
3	1
5	1
6	1

B	
instance_id	offset_list
1	2
2	2
3	2
4	1
5	2

G	
instance_id	offset_list
1	4
4	4

(b) Numerical dataset DB_N

Attribute : $attr_1$							
V = [0.27, 0.37, 0.38, 0.4], δ = 0.2							
{($attr_1$,[0.25,0.29])}		{($attr_1$,[0.35,0.39])}		{($attr_1$,[0.36,0.40])}		{($attr_1$,[0.38,0.42])}	
instance_id	offset_list	instance_id	offset_list	instance_id	offset_list	instance_id	offset_list
6	1	3	1	1	1	1	1
		5	1	3	1	3	1
				5	1		

(c) Interval dataset D_I

{L}	
instance_id	offset_list
1	1
2	1
3	1
5	1

{K}	
instance_id	offset_list
1	2
2	3
3	3
4	1
5	2

{N}	
instance_id	offset_list
6	1

(d) Time series dataset DB_T

motif id	time series motif (δ = 2)
m_1	{T_{11}[1,3], T_{31}[1,3], T_{61}[2,4]}
m_2	{T_{11}[1,3], T_{21}[2,4]}
m_3	{T_{12}[1,3], T_{22}[2,4], T_{32}[2,4]}
m_4	{T_{12}[2,4], T_{32}[2,4]}
m_5	{T_{42}[1,3], T_{52}[3,5]}

Table 5.3. Generation of length 1 frequent patterns involving single kind of data

(a) Transformed dataset DB'_T

Instance Id	Interval Motif Data	Class
1	(m_1,1,3),(m_2,1,3),(m_3,1,3),(m_4,2,4)	1
2	(m_2,2,4),(m_3,2,4)	1
3	(m_1,1,3),(m_3,2,4),(m_4,2,4)	1
4	(m_5,1,3)	1
5	(m_5,3,5)	0
6	(m_1,2,4)	0

(b) Index m_1

motif : m_1	
instance_id	offset_list
1	1
3	1
6	1

Table 5.4. Transformed dataset DB'_T and index of motif m_1

its frequency and record the instance id and its offset in the index. Table 5.3(b) lists the length 1 patterns generated using attribute $attr_1$.

For **interval data**, a length 1 temporal pattern is simply a frequent event. We obtain the frequency of events and record the instance ids and their offsets of those instances that support these frequent events.

For **time series data**, a length 1 pattern is a frequent motif. Hence, we first discover all frequent time series motifs using method described in previous chapter. Each discovered frequent motif is given a unique id. For example, Table 5.3(d) lists the discovered five motifs. Here, motif m_1 is a set of three subsequences from time series of attribute 1(i.e., time series 1). Note that, T_{61}[2,4] denotes a subsequence from time points 2 to 4 from time series 1 of instance 6. Based on these frequent motifs, we transform the time series data DB_T into an interval-based dataset DB'_T as follows: we add motif m with its start and end time to the i^{th} entry in DB'_T if m has a subsequence from instance i. For example, motif m_1 has one subsequence starting from time points 2 to 4 in time series 1 of instance 6, hence we append (m_1,2,4) to instance 6 of DB'_T. Similarly, we append (m_1,1,3) in instance 3 of DB'_T. Table 5.4 shows the result of such a transformation. Note that, motifs in each instance of DB'_T are ordered with respect to start time, end time and motif id. Now, DB'_T is processed similarly as interval data DB_I.

Once the length 1 patterns for the given data have been generated, Algorithm MineSingle generates a new candidate pattern by first trying to **extend** an existing pattern α(Line 6). We put all extended patterns in $frePatSet$ for further extension and annotation. The algorithm also tries to **annotate** the existing pattern to form new patterns if it is from interval or time series data(Line 8). We put all extended patterns in $patSet$(Line 9). The process is repeated till no pattern is left in $frePatSet$. Finally, the set of generated patterns is $patSet$(Line 11). We will illustrate the extend and annotate process with examples in the following subsections.

Extend Pattern

Consider the extension of **itemset** $\{$A,B$\}$. Table 5.5(a) shows an index of the itemset $\{$A,B$\}$ w.r.t. the dataset DB_C in Table 5.5(b). The first entry $(1, \{1, 2\})$ indicates that item A is at offset 1 and item B is at offset 2 in instance 1. Any item appearing after the offset 2 in instance 1 is a potential candidate item for extending the itemset. In this case, we extend the pattern $\{$A, B$\}$ with item E and G to form candidate patterns $\{$A, B, E$\}$ and $\{$A, B, G$\}$ respectively. Note that, for each candidate pattern, we maintain the instance ids from which the pattern is generated. Similarly, other entries from index of the itemset $\{$A,B$\}$ are processed. Table 5.5(c) lists the candidate patterns generated. Finally, candidate patterns which are frequent are indexed and returned.

Consider the extension of **numerical pattern** $\{(attr_1$, [0.36,0.40])$\}$. Table 5.6(a) shows the index of a length 1 numerical pattern $\{(attr_1$, [0.36,0.40])$\}$ w.r.t the dataset DB_N in Table 5.6(b). The first entry $(1, \{1\})$ indicates that attribute $attr_1$ is at offset 1 in instance 1. Any attribute appearing after the offset 1 in instance 1 is a potential candidate for extension. From the index, we access only those instances that supports $\{(attr_1$,[0.36,0.40])$\}$ in order to obtain the possible values for each candidate for extension. In this case, there is only one possible value for $attr_2$ = $\{$0.6$\}$ and

(a) Index

itemset : {A,B}	
instance_id	offset_list
1	1,2
2	1,2
3	1,2
5	1,2

(b) Dataset DB_C

Instance id	Categorical Data	Class
1	A,B,E,G	1
2	A,B,D	1
3	A,B,D	1
4	B,D,E,G	1
5	A,B,D	0
6	A,D,E	0

(c) Candidate patterns

itemsets	set of instance ids
{A,B,D}	{2,3,5}
{A,B,E}	{1}
{A,B,G}	{1}

Table 5.5. Example extension of itemset {A,B}

two possible values for $attr_3$ = {0.5,0.6}. As a result, much savings can be achieved with only three candidates generated as shown in Table 5.6(c). For each candidate pattern, we obtain the supporting instances. Similarly, other entries from index of the numerical pattern {($attr_1$, [0.36,0.40])} are processed. Finally, candidate patterns which are frequent are indexed and returned.

(a) Index

{($attr_1$,[0.36,0.40])}	
instance_id	offset_list
1	1
3	1
5	1

(b) Dataset DB_N

Instance Id	Numerical Data	Class
1	($attr_1$,0.4),($attr_2$,0.6)	1
2	($attr_2$,0.8),($attr_3$,0.2)	1
3	($attr_1$,0.38),($attr_3$,0.5)	1
4	($attr_2$,0.5),($attr_3$,0.9)	1
5	($attr_1$,0.37),($attr_3$,0.6)	0
6	($attr_1$,0.27),($attr_2$,0.4)	0

(c) Candidate patterns

numerical patterns	set of instance ids
{($attr_1$,[0.36,0.40]),($attr_2$,[0.4,0.8])}	{1}
{($attr_1$,[0.36,0.40]),($attr_3$,[0.3,0.7])}	{3,5}
{($attr_1$,[0.36,0.40]),($attr_3$,[0.4,0.8])}	{3,5}

Table 5.6. Example extension of numerical pattern {($attr_1$,[0.36,0.40])}

Similarly, we can extend the patterns from interval and time series data. Suppose we have a **temporal pattern** ip = {L $\overset{Overlap[0,0,0,1,0]}{\longrightarrow}$ K}. Table 5.7(a) shows the $index$ of this pattern with respect to the dataset DB_I in Table 5.7(b). The first entry in the index $(1, \{1, 2\})$ indicates that event L is at offset 1 and event K is at offset 2 of instance 1. From dataset DB_I, we notice that there are two events, S and T, that appear after offset 2 in instance 1. Hence we can extend ip by events S and T respectively. First, we attempt to extend ip by event S. We determine S's temporal relationship with the events in ip. From the start and end times, we observe that L is "Before" S and K "Meets" S. Hence, the generated pattern for this event is $\{L \overset{Overlap[0,0,0,1,0]}{\longrightarrow} K \overset{Meet[0,0,1,0,0]}{\longrightarrow} S\}$. Similarly, we process event T and obtain $\{L \overset{Overlap[0,0,0,1,0]}{\longrightarrow} K \overset{Before[0,0,0,0,0]}{\longrightarrow} T\}$. All other entries from index of temporal pattern {L $\overset{Overlap[0,0,0,1,0]}{\longrightarrow}$ K} are processed similarly. Table 5.7(c) lists the generated candidate patterns and their support instance ids. The candidate patterns which are frequent are indexed and returned. It is possible that, an instance supports temporal pattern multiple times. In such situation, we have indexed all occurrences of the temporal patterns.

(a) Index

{L $\overset{Overlap[0,0,0,1,0]}{\longrightarrow}$ K}	
instance_id	offset_list
1	1,2
2	1,3
3	1,3
5	1,2

(b) Dataset DB_I

Instance Id	Interval Event Data	Class
1	(L,1,6),(K,4,8),(S,8,10),(T,11,12)	1
2	(L,1,5),(T,2,7),(K,3,9),(S,9,13)	1
3	(L,3,8),(T,3,9),(K,5,10),(S,10,12)	1
4	(K,2,4),(S,6,10)	1
5	(L,1,6),(K,3,8),(S,4,10)	0
6	(N,5,10)	0

(c) Candidate patterns

temporal patterns	set of instance ids
{L $\overset{Overlap[0,0,0,1,0]}{\longrightarrow}$ K $\overset{Meet[0,0,1,0,0]}{\longrightarrow}$ S}	{1,2,3}
{L $\overset{Overlap[0,0,0,1,0]}{\longrightarrow}$ K $\overset{Before[0,0,0,0,0]}{\longrightarrow}$ T}	{1}
{L $\overset{Overlap[0,0,0,1,0]}{\longrightarrow}$ K $\overset{Overlap[0,0,0,2,0]}{\longrightarrow}$ S}	{5}

Table 5.7. Example extension of temporal pattern {L $\overset{Overlap[0,0,0,1,0]}{\longrightarrow}$ K}

Annotate Pattern

In order to generate a new pattern via annotation, we need to find the frequent time tag information between adjacent pairs of events/motifs in a given pattern α. This is obtained by calculating the differences between start times of adjacent pair of events E_{i+1} and E_i in each instance present in index of α. This difference is known as time lag between E_i and E_{i+1} and denoted as $lag_{i,i+1}$. For each time lag $lag_{i,i+1}$ between E_i and E_{i+1}, we obtain all instances in $index$ that supports $lag_{i,i+1}$ between E_i and E_{i+1}. With this, we can generate the frequent annotated patterns.

For example, given a temporal pattern tp = $\{L \overset{Overlap[0,0,0,1,0]}{\longrightarrow} K\}$ and its index in Table 5.7(a). From the index, we know that the difference between the start times of events K and L is either 2 or 3. Thus, we have two candidate annotated patterns $\{L \overset{Overlap[0,0,0,1,0](2)}{\longrightarrow} K\}$ and $\{L \overset{Overlap[0,0,0,1,0](3)}{\longrightarrow} K\}$. For both candidate patterns, we obtain instances from index of tp that supports them.

Similarly, if given pattern is tp = $\{L \overset{Overlap[0,0,0,1,0]}{\longrightarrow} K \overset{Meet[0,0,1,0,0]}{\longrightarrow} T\}$, then we obtain possible time lag between event L and K as well as between event K and T using index of tp. Assume, time lag between L and K is 2 or 3 and time lag between K and T is 4. Then, we generate two candidate annotated patterns $\{L \overset{Overlap[0,0,0,1,0](2)}{\longrightarrow} K \overset{Meet[0,0,1,0,0](4)}{\longrightarrow} T\}$ and $\{L \overset{Overlap[0,0,0,1,0](3)}{\longrightarrow} K \overset{Meet[0,0,1,0,0](4)}{\longrightarrow} T\}$.

Note that, HT pattern $\{L \overset{Overlap[0,0,0,1,0](2)}{\longrightarrow} K \overset{Meet[0,0,1,0,0](4)}{\longrightarrow} T\}$ can be generated by extending $\{L \overset{Overlap[0,0,0,1,0](2)}{\longrightarrow} K\}$ or annotating $\{L \overset{Overlap[0,0,0,1,0]}{\longrightarrow} K \overset{Meet[0,0,1,0,0]}{\longrightarrow} T\}$. Our approach selects the second alternative as if $\{L \overset{Overlap[0,0,0,1,0]}{\longrightarrow} K \overset{Meet[0,0,1,0,0]}{\longrightarrow} T\}$ is not frequent then we do not need to annotate it even $\{L \overset{Overlap[0,0,0,1,0](2)}{\longrightarrow} K\}$ is frequent. This is a reason why annotated pattern is not used for extension and annotation(Lines 8-9 in Algorithm 11).

5.2.2 Algorithm MineMultiple

Now, we systematically explore the search space to generate heteorogeneous patterns (HT patterns) involving multiple kinds of data. Algorithm MineMultiple (see Algorithm 12) uses the patterns generated from Algorithm MineSingle to grow the heterogenous pattern α in three dimensions:

- **Extend** α. A new HT pattern is formed by increasing the length of the rightmost non-empty pattern of α by 1. For example, if α = [$\{A, D\}$, $\emptyset$, $\{L\}$, $\emptyset$]. Then, we select the rightmost non-empty pattern $\{$L$\}$ for extension. Here, [$\{$A,D$\}$,$\emptyset$, $\{L \overset{Overlap[0,0,0,1,0]}{\longrightarrow} T\}$, $\emptyset$] is a possible extension of [$\{A, D\}$, $\emptyset$, $\{L\}$, $\emptyset$]. Table 5.10(c) lists the new HT patterns generated when α is extended. Similarly, if α = [$\{A, D\}$, $\emptyset$, $\emptyset$, $\{m_1\}$]. Then, $\{m_1\}$ is selected for extension. Here, [$\{A, D\}$, $\emptyset$, $\emptyset$, $\{m_1 \rightarrow m_2\}$] is a possible extension of α.

- **Annotate** α. A new HT pattern is formed by annotating the rightmost non-empty pattern of α. Note that, only patterns from interval or time series pattern are required annotation. For example, if α = [$\{$A,D$\}$,$\emptyset$,$\{L \overset{Overlap[0,0,0,1,0]}{\longrightarrow} K\}$,$\emptyset$], the example of annotated pattern is [$\{$A,D$\}$,$\emptyset$,$\{L \overset{Overlap[0,0,0,1,0](2)}{\longrightarrow} K\}$,$\emptyset$]. But, no annotation is required if α = [$\{A$,$D\}$, $\{$D,[3,6]$\}$, $\emptyset$, $\emptyset$] as both patterns from interval and time series are empty.

- **Enlarge** α. A new HT pattern is formed by increasing the size of HT pattern α by 1. In this chapter, all empty patterns that appears after the right most non-empty pattern in α are candidates for enlargement. For example, if α = ($\{$A,D$\}$, $\emptyset$, $\emptyset$, $\emptyset$), then we can increase the size of α by finding length 1 frequent patterns from either numerical, interval, or time series data.

Algorithm 12 MineMultiple(DB, $minsup$)

Input : DB, $minsup$

Output : Set of HT patterns $patSet$

1: HT pattern α = [$\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$]
2: $patSet$ = $\emptyset$
3: $mdPatSet$ = {**enlarge** pattern α}
4: **while** $mdPatSet \neq \emptyset$ **do**
5: Select pattern α from $mdPatSet$ and remove it from $mdPatSet$
6: Store α in $patSet$
7: $enlPatSet$ = {**enlarge** pattern α}
8: $mdPatSet$ = $mdPatSet \cup enlPatSet$
9: $extPatSet$ = {**extend** pattern α}
10: $mdPatSet$ = $mdPatSet \cup extPatSet$
11: $annPatSet$ = {**annotate** pattern α}
12: $patSet$ = $patSet \cup annPatSet$
13: **for** each annotated pattern $\beta \in annPatSet$ **do**
14: $enlPatSet$ = {**enlarge** pattern β}
15: $mdPatSet$ = $mdPatSet \cup enlPatSet$
16: **end for**
17: **end while**
18: return $patSet$

Similarly, if α = ({A,D}, $\emptyset$, $\{L\}$, $\emptyset$), then we enlarge α by finding length 1 pattern from time series data(i.e., motif).

Algorithm 12 starts with an initial HT pattern α = [$\emptyset$, $\emptyset$, $\emptyset$, $\emptyset$] (Line 1). Note that, the size of α is 0. In line 2, we **enlarge** α by increasing its size by 1. This yields a set of new HT patterns where length of non-empty pattern of each generated pattern is 1. Briefly, this step generates one HT pattern for each length 1 frequent patterns of single data kind. Note that, the index of HT pattern α is the index of right most non-empty pattern in α. All generated patterns are stored in $mdPatSet$(Line 2). Next, for each HT pattern α in $mdPatSet$, we generate new patterns by enlarging (Line 7), extending (Line 9) and annotating (Line 11). This algorithm terminates when $mdPatSet$ is empty. Now, we illustrate the enlarge, extend and annotate process for HT patterns with examples.

Enlarge HT Pattern

Consider the **enlargement** of HT pattern α = [$\{A, D\}$, $\emptyset$, $\emptyset$, $\emptyset$]. We can increase the size of α by finding length 1 frequent patterns from either numerical, interval, or time series data such that the combination with itemset $\{$A,D$\}$ are frequent. we observe that α is present in instances 2, 3, 5 and 6. Hence, we limit the search to only these 4 instances to find frequent numerical, interval, or time series patterns of length 1 while enlarging pattern α. Finding length 1 frequent pattern from each kind of data is explained in section 5.2.1. Table 5.8(c) and Table 5.9(c) lists the new HT patterns obtained using numerical and interval data respectively. Finally, all generated frequent HT patterns are indexed and returned.

(a) Index

[{A,D},∅,∅,∅]	
instance_id	offset_list
2	1,3
3	1,3
5	1,3
6	1,2

(b) Dataset DB_N

Instance Id	Numerical Data	Class
1	($attr_1$,0.4),($attr_2$,0.6)	1
2	($attr_2$,0.8),($attr_3$,0.2)	1
3	($attr_1$,0.38),($attr_3$,0.5)	1
4	($attr_2$,0.5),($attr_3$,0.9)	1
5	($attr_1$,0.37),($attr_3$,0.6)	0
6	($attr_1$,0.27),($attr_2$,0.4)	0

(c) Candidate patterns

HT pattern	set of instance ids
[{A,D},{($attr_1, [0.35, 0.39]$)},∅,∅]	{3,5}
[{A,D},{($attr_1, [0.36, 0.40]$)},∅,∅]	{3,5}
[{A,D},{($attr_1, [0.25, 0.29]$)},∅,∅]	{6}
[{A,D},{($attr_2, [0.6, 1.0]$)},∅,∅]	{2}
...	...

Table 5.8. Example enlargement of HT pattern : [{A,D},∅,∅,∅] using numerical data

(a) Index

[{A,D},∅,∅,∅]	
instance_id	offset_list
2	1,3
3	1,3
5	1,3
6	1,2

(b) Dataset DB_I

Instance Id	Event List	Class
1	(L,1,6),(K,4,8),(S,8,10),(T,11,12)	1
2	(L,1,5),(T,2,7),(K,3,9),(S,9,13)	1
3	(L,3,8),(T,3,9),(K,5,10),(S,10,12)	1
4	(K,2,4),(S,6,10)	1
5	(L,1,6),(K,3,8),(S,4,10)	0
6	(N,5,10)	0

(c) Candidate patterns

HT patterns	set of instance ids
[{A,D},∅,{L},∅]	{2,3,5}
[{A,D},∅,{K},∅]	{2,3,5}
[{A,D},∅,{S},∅]	{2,3,5}
[{A,D},∅,{T},∅]	{2,3}
[{A,D},∅,{N},∅]	{6}

Table 5.9. Example enlargement of HT pattern : [{A,D},∅,∅,∅] using third empty pattern

Extend HT Pattern

Consider **extension** of HT pattern α = [{A, D}, ∅, {L}, ∅]. As mentioned earlier, we select the rightmost non-empty pattern, {L}, for extension. We observe that α is present in instances 2, 3 and 5. Hence, we limit the search to only these 3 instances to find extension of pattern {L}. The first entry in the index (2,{1}) indicates that event L is at offset 1 in instance 2′s interval data. From dataset DB_I, we notice that there are three events, T, K, and S, that appear after offset 1 in instance 2. We determine T's temporal relationship with L and generate [{A,D},∅,{$L \xrightarrow{Overlap[0,0,0,1,0]} T$},∅]. Similarly, events K and S are processed. Next, all entries from index are processed similarly. Table 5.10(c) lists extended patterns of α. Finally all generated frequent HT patterns are indexed and returned.

(a) Index

[{A,D},∅,{L},∅]	
instance_id	offset_list
2	1
3	1
5	1

(b) Dataset DB_I

Instance Id	Event List	Class
1	(L,1,6),(K,4,8),(S,8,10),(T,11,12)	1
2	(L,1,5),(T,2,7),(K,3,9),(S,9,13)	1
3	(L,3,8),(T,3,9),(K,5,10),(S,10,12)	1
4	(K,2,4),(S,6,10)	1
5	(L,1,6),(K,3,8),(S,4,10)	0
6	(N,5,10)	0

(c) Candidate patterns

HT patterns	set of instance ids
[{A,D},∅,{$L \overset{Overlap[0,0,0,1,0]}{\longrightarrow} T$},∅]	{2}
[{A,D},∅,{$L \overset{Overlap[0,0,0,1,0]}{\longrightarrow} K$},∅]	{2,3,5}
[{A,D},∅,{$L \overset{Before[0,0,0,0,0]}{\longrightarrow} S$},∅]	{2,3}
...	...

Table 5.10. Example extension of HT pattern : [{A,D},∅,{L},∅]

Annotate HT Pattern

Consider **annotation** of HT pattern α = [{A, D}, ∅, {{$L \overset{Overlap[0,0,0,1,0]}{\longrightarrow} K$}}, ∅]. As mentioned earlier, we select the rightmost non-empty pattern, {$L \overset{Overlap[0,0,0,1,0]}{\longrightarrow} K$}, for annotation. We observe that α is present in instances 2, 3 and 5 and the difference between the start times of events K and L is 2 in these instances. Hence, only one annotated pattern, [{A, D}, ∅, {{$L \overset{Overlap[0,0,0,1,0](2)}{\longrightarrow} K$}}, ∅], is generated.

Optimizations

We have devise four optimization strategies to further reduce the total number of HT patterns to be formed.

- **Optimization 1:** Suppose a HT pattern α = [cp,np,∅,∅] is obtained by **enlarging** an existing HT pattern [cp,∅,∅,∅] using length 1 pattern np from numerical data. Let sup(np) be the

support of pattern np obtained while mining numerical data in the first stage. If sup(α) = sup(np), then patterns α and np are generated from the same set of instances. In other words, pattern α and pattern [$\emptyset$,n_p,$\emptyset$,$\emptyset$] are closed patterns. Hence, we do not need to process α further, as all heterogenous patterns that can be generated using α, can be generating using [$\emptyset$,n_p,$\emptyset$,$\emptyset$]. An example of this is the second pattern in Table 5.8(c). Its support is equal to support of pattern [$\emptyset$,{($attr_1[0.35, 0.39]$)},$\emptyset$,$\emptyset$]. Note that, $patternSet_N$ maintains sup(np).

- **Optimization 2:** Suppose a HT pattern α = [cp,np,$\emptyset$,$\emptyset$] is obtained by **extending** an existing HT pattern β = [cp,np',$\emptyset$,$\emptyset$]. Let sup(np) be the support of pattern np obtained while mining numerical data in the first stage. If sup(α) = sup(np), then patterns α and np are generated from the same set of instances. In other words, pattern α and pattern [$\emptyset$,np,$\emptyset$,$\emptyset$] are closed patterns. Hence, we do not need to process α further, as all patterns that can be generated using α, can be generated using [$\emptyset$,np,$\emptyset$,$\emptyset$].

- **Optimization 3:** This optimization is based on the observation that if no frequent HT pattern is generated when enlarging an existing HT pattern α (i.e., $enlPatSet$ = $\emptyset$), then any extended or annotated patterns of α will not generate frequent HT patterns if it is enlarged. For example, let α = [$\{A, D\}$, $\emptyset$, $\{L\}$, $\emptyset$]. If no frequent pattern is generated when α is enlarged, then enlargement of [$\{A, D\}$, $\emptyset$, $\{L \xrightarrow{Overlap[0,0,0,1,0]} K\}$, $\emptyset$] will not generate any frequent patterns. Thus, we will not enlarge any extended or annotated patterns of α.

- **Optimization 4:** This optimization is based on the observation that if annotation of pattern involving single kinds of data does not exist in the first stage, then we do not need to annotate its corresponding HT pattern. For example, if annotation of pattern α_1 = $\{L \xrightarrow{Overlap[0,0,0,1,0]} K\}$ does not generate any pattern in the first stage, then annotation of α_2 = [$\{A, B\}$, $\emptyset$, $\{L$

$\xrightarrow{Overlap[0,0,0,1,0]} K\}, \emptyset]$ will not generate any pattern in second stage. Thus, we do not annotate pattern α_2.

5.3 Algorithm HTClassifier

Frequent patterns reflect strong associations between items and carry the underlying semantics of the data. Thus, they are potentially useful features for classification. However, due to explosive nature of frequent pattern mining at low support threshold, the frequent pattern-based feature construction for classification could encounter a computational bottleneck. Thus, we design an efficient mining strategy, called HTClassifier, which directly mines the discriminative patterns without generating the whole set of features. HTClassifier modifies HTMiner to generate a set of essential heterogenous patterns for classification(see Algorithm 13).

Algorithm 13 HTClassifier

Input : DB, $minsup$, max_conf, min_conf
Output : A set of essential patterns $essPatSet$

1: $essPatSet$ = $\emptyset$;
2: **while** true **do**
3: /* Find best essential pattern */
4: α = **FindEssentialPattern(**DB**,** $minsup$**,** max_conf**,** min_conf**)**
5: **if** α = ϕ **then**
6: break
7: **end if**
8: $essPatSet$ = $essPatSet \cup \alpha$
9: /* Instance elimination */
10: DB = DB - DB_α
11: **if** DB = ϕ **then**
12: break
13: **end if**
14: **end while**
15: return $essPatSet$

Algorithm 14 FindEssentialPattern(DB, $minsup$, max_conf, min_conf)

Output : Essential pattern $essPat$

Global variable : $essPat$ = $\emptyset$, N = 4, $patternSet$ = $\emptyset$

1: **for** p = 1 to N **do**
2: DB_p = projected database of DB for p^{th} kind of data
3: [$patternSet_p$, s] = MineEssentialSingle(DB_p, $minsup$, max_conf, min_conf)
4: **if** $s > minsup$ **then**
5: $minsup$ = s
6: **end if**
7: **end for**
8: Remove infrequent attributes from DB
9: MineEssentialMultiple(DB, $minsup$, max_conf, min_conf)
10: return $essPat$

Given $minsup$, max_conf, min_conf and database DB, we adopt a sequential coverage approach [61] to generate a set of essential HT patterns. At each iteration, an essential heterogenous pattern, α, is discovered from the dataset with multiple kinds of data DB by calling FindEssential-Pattern (Line 4). All the instances that support α, denoted as DB_α, are eliminated from DB (Line 10) thus reducing the size of the dataset for subsequent iterations. This process is repeated till no essential pattern is generated (Line 6) or all instances have been eliminated (Line 12). Finally, the set of discovered essential patterns $essPatSet$ is returned (Line 15).

FindEssentialPattern (see Algorithm 14) is similar to HTMiner with additional optimizations. FindEssentialPattern first projects the dataset DB into the respective projected databases based on the kinds of data that exists in DB (Line 2). It then calls MineEssentialSingle for each projected databases. This procedure generates a set of frequent patterns $patternSet$ along with their support values(Line 3). It also discovers an essential pattern if any. If we find more than one essential patterns, the pattern with the highest support is selected as an essential pattern. Its support value, s, is compared against $minsup$. If s greater than $minsup$, we update the $minsup$ to s (Lines 4-5). This allows subsequent mining to target only those patterns whose support is greater

than s, thus reducing the risk of overfitting in classification. By successively raising the *minsup* threshold, we are able to prune off a significant number of patterns while mining remaining data. Details of MineEssentialSingle are given in Section 5.3.1. Lines 9-11 invoke MineEssentialMultiple to generate heterogenous patterns. Details are given in Section 5.3.2.

5.3.1 Algorithm MineEssentialSingle

Algorithm MineEssentialSingle is similar to algorithm MineSingle with additional constraints(Lines 8-12 and Lines 16-20) . Initially, Algorithm MineEssentialSingle (see Algorithm 15) obtains all the frequent patterns of length 1 from the given dataset with single kind of data DB_p (Line 2). Once the length 1 patterns for the given data has been generated, Algorithm MineEssentialSingle generates a new candidate pattern by first trying to extend an existing pattern α(Line 6). If any pattern β from the generated patterns $expPatSet$ is an essential pattern, we update $minsup$ and $essPat$ (Line 9). Otherwise, we put β in $frePatSet$ for further extension. The algorithm also tries to annotate the existing pattern to form new patterns if it is from interval or time series data(Line 14). If any pattern β from the generated patterns $expPatSet$ is an essential pattern, we update $minsup$ and $essPat$ (Line 17). Otherwise, we put β in $patSet$ (Line 19). The process is repeated till no pattern is left in $frePatSet$. Finally, the set of generated patterns $patSet$ and updated $minsup$ are returned (Line 23). Note that, $minsup$ is only updated if any essential pattern is generated. The extend and annotate process are same as explained earlier.

Algorithm 15 MineEssentialSingle(DB_p, $minsup$, max_conf, min_conf)

Output : A frequent patterns $patSet$, updated $minsup$

```
1:  patSet = ∅    // set of generated patterns
2:  frePatSet = {length 1 frequent patterns from DB_p}
3:  while frePatSet ≠ ∅ do
4:     Select pattern α from frePatSet and remove it from frePatSet
5:     Add α to patSet
6:     extPatSet = {extend pattern α}
7:     for each generated pattern β ∈ extPatSet do
8:        if β is an essential pattern then
9:           Set (minsup = sup(β) and essPat = β)
10:       else
11:          Add β to frePatSet
12:       end if
13:    end for
14:    annPatSet = {annotate pattern α}
15:    for each annotated pattern β ∈ annPatSet do
16:       if β is an essential pattern then
17:          Set (minsup = sup(β) and essPat = β)
18:       else
19:          Add β to patSet
20:       end if
21:    end for
22: end while
23: return {patSet, minsup}
```

5.3.2 Algorithm MineEssentialMultiple

Algorithm 16 starts with an initial HT pattern α = [∅, ∅, ∅, ∅] (Line 1). Note that, the size of α is 0. In line 2, we **enlarge** α by increasing its size by 1. This yields a set of new HT patterns where length of non-empty pattern of each generated pattern is 1. Briefly, this step generates one HT pattern for each length 1 frequent patterns of single data kind. All generated patterns are stored in $mdPatSet$(Line 2). Next, for each HT pattern α in $mdPatSet$, we generate new patterns by enlarging (Lines 5- 12), extending (Lines 13-20) and annotating (Lines 21-35). If the new pattern is essential, we update $minsup$ and $essPat$, otherwise we process further. This algorithm terminates when $mdPatSet$ is empty.

Algorithm 16 MineEssentialMultiple(DB, $minsup$, max_conf, min_conf)

```
1: HT pattern α = [∅, ∅, ∅, ∅]
2: mdPatSet = {enlarge pattern α}
3: while mdPatSet ≠ ∅ do
4:    Select pattern α from mdPatSet and remove it from mdPatSet
5:    enlPatSet = {enlarge pattern α}
6:    for each enlarged pattern β ∈ enlPatSet do
7:       if β is an essential pattern then
8:          Set (minsup = sup(β) and essPat = β)
9:       else
10:         Add β to mdPatSet
11:      end if
12:   end for
13:   extPatSet = {extend pattern α}
14:   for each extended pattern β ∈ extPatSet do
15:      if β is an essential pattern then
16:         Set (minsup = sup(β) and essPat = β)
17:      else
18:         Add β to mdPatSet
19:      end if
20:   end for
21:   annPatSet = {annotate pattern α}
22:   for each annotated pattern β ∈ annPatSet do
23:      if β is an essential pattern then
24:         Set (minsup = sup(β) and essPat = β)
25:      else
26:         enlPatSet = {enlarge pattern β}
27:         for each enlarged pattern γ ∈ enlPatSet do
28:            if γ is an essential pattern then
29:               Set (minsup = sup(γ) and essPat = γ)
30:            else
31:               Add γ to mdPatSet
32:            end if
33:         end for
34:      end if
35:   end for
36: end while
```

We devise an additional optimization strategy to further reduce the total number of candidate HT patterns. This optimization is based on the upper bound estimation of confidence and is used for early termination. Given a HT pattern α with $|\alpha_c|$ number of instances that support α with class label c. Let $|DB|$ be the number of instances in a dataset DB. If α is frequent but not discriminative, then it needs to be extended, enlarged or annotated only if the following conditions are satisfied:

1. $|\alpha_c| \geq min\_sup * |DB|$, and

2. $\frac{|\alpha_c|}{minsup*|DB|} \geq max\_conf$, and

3. $\frac{|\alpha_{c'}|}{sup(\alpha)} \leq min\_conf \quad \forall c' \neq c$

Proof: In the proof, we need to show that if α does not satisfy the three conditions, then any HT pattern, say β, that is generated by extending, enlarging or annotating from α cannot be essential.

Case I. β is infrequent.

In this case, we are done since β cannot be an essential HT pattern.

Case II. β is frequent.

Since β is frequent, we have $sup(\beta) > min\_sup$. We know $|\beta_c| \leq |\alpha_c|$ as β is a longer pattern than α and $\frac{|\alpha_c|}{min\_sup*|DB|} < max\_conf$ since α is not essential. This implies:

$$\begin{aligned} conf(\beta) &= \frac{|\beta_c|}{sup(\beta)} \leq \frac{|\beta_c|}{min\_sup * |DB|} \\ &\leq \frac{|\alpha_c|}{min\_sup * |DB|} < max\_conf \end{aligned}$$

5.4 Experimental Study

We present the results of experiments to examine the efficiency of HTMiner and effectiveness of the HT patterns discovered by HTClassifier for classification. Experiments are carried out on an Intel Core Duo E6550 with 3.25GB of main memory running Windows XP Professional. We implemented all algorithms in Visual C#. In particular, we implement four algorithms named HTMiner, HTMiner+, HTClassifier, and HTClassifier+. Table 5.11 lists the basic algorithm with description. Algorithms with suffix "+" incorporates the optimizations described in the respective sections.

Algorithm	**Purpose**
HTMiner	Mining complete set of frequent patterns
HTClassifier	Mining set of essential patterns

Table 5.11. Algorithm Description

We use two real world datasets, Hepatitis and Stulong, from the 2004 ECML Discovery Challenge[1]. The Hepatitis dataset captures the data of patients with Hepatitis B or Hepatitis C. In general, a biopsy is needed to confirm the type of hepatitis, however biopsy is both invasive and costly. Hence, predicting the type of hepatitis based on only laboratory results will be highly beneficial. The Stulong dataset describes the risk factors of middle aged men that may lead to atherosclerosis cardiovascular disease. Table 5.12 shows the characteristics of these datasets.

Baseline Algorithm. We devise a two-phase baseline algorithm for our comparative studies. The first phase utilizes existing algorithms to mine patterns from each kind of data separately. The second phase generates candidate HT patterns by enumerating the possible combinations of patterns obtained in the first phase. The supports and confidences of the candidate HT patterns are

Data Type	Characteristics	DataSet	
		Hepatitis	**Stulong**
DB_C	Avg. size of categorical item set	25	20
DB_N	Avg. size of numerical item set	100	40
DB_I	Avg. number of events	27	20
DB_T	Avg. number of time series	8	10
	Avg. length of time series	22	12
Total Instances(Patients)		459	855
Class Distribution		[HepB = 185] [HepC = 275]	[non-CVD = 583] [CVD = 272]

Table 5.12. Dataset Description

obtained by scanning the original dataset. To obtain a set of essential patterns, we iteratively select an essential HT pattern and remove all the instances that are correctly supported by the selected pattern. The process terminates when all instances have been eliminated. Note that this baseline algorithm generates exactly the same set of essential HT patterns as the optimized algorithms.

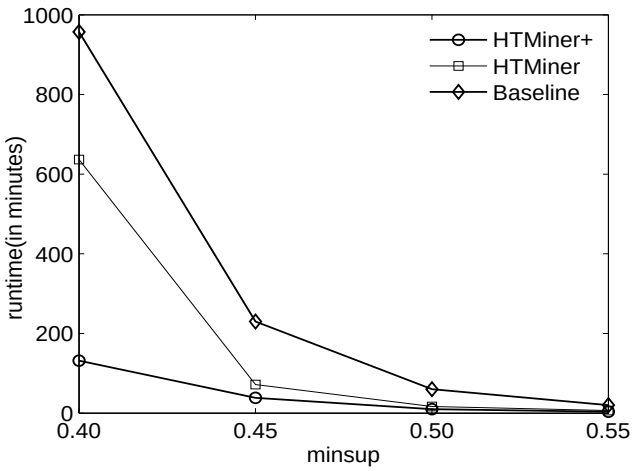
5.4.1 Efficiency Experiments

First, we explain the default parameters used in all experiments. We set the similarity threshold of numerical attribute at 10% of its standard deviation, and the time lag similarity threshold to 1 days. The length of a time series motif ranges from 5 to 7 and its similarity threshold is 2.

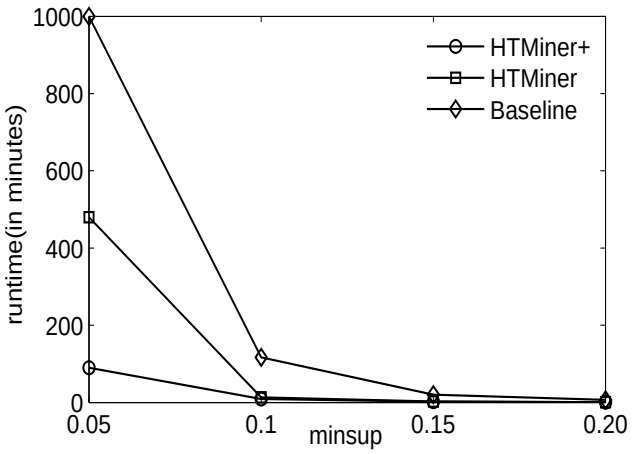
Mining complete set of frequent HT patterns

Our first set of experiments compare efficiency of mining complete set of heterogenous patterns, i.e., HTMiner, HTMiner+ and Baseline. We examine the effect of varying the minimum support threshold for both datasets. We observed that, HTMiner+ is efficient algorithms. As we

reduce minimum support threshold, the running time of all algorithms increase drastically. This is obvious, as time complexity of frequent pattern mining algorithm is exponential. However, proposed optimization and prefix based indexing enable us to discover patterns efficiently.



(a) Hepatitis Dataset

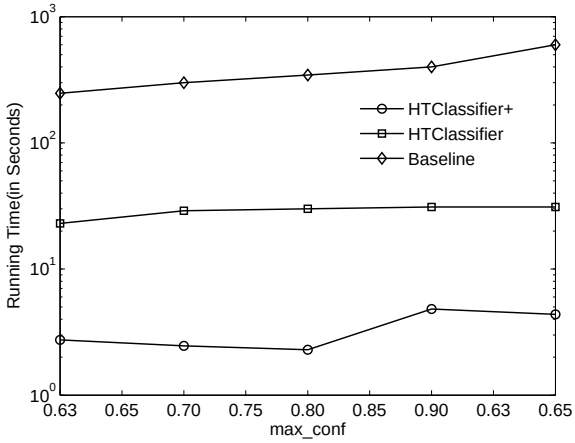


(b) Stulong Dataset

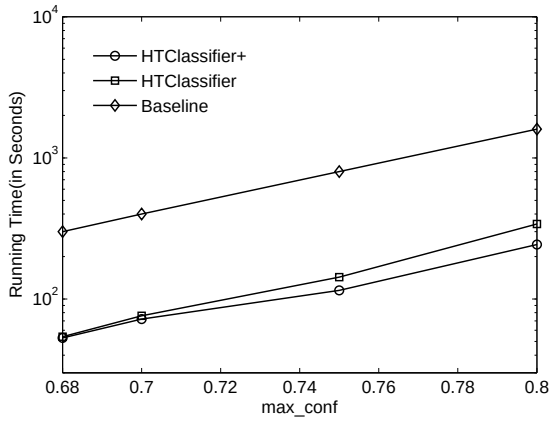
Figure 5.1. Effect of varying $minsup$

Mining set of essential patterns for classification

Our second set of experiments compare efficiency of mining set of essential heterogenous patterns. We examine the effect of the parameters max_conf and $minsup$ on the performance of the algorithms HTClassifier, HTClassifier+ and Baseline.



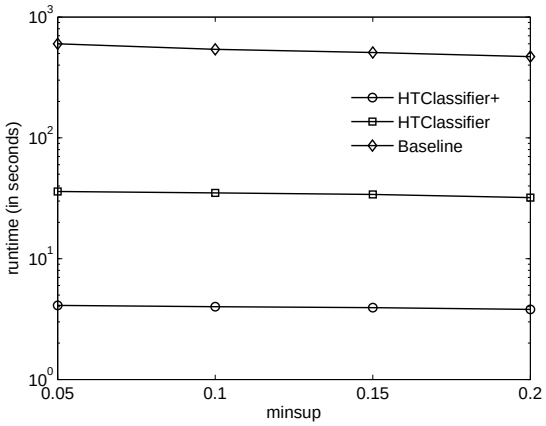
(a) Hepatitis Dataset



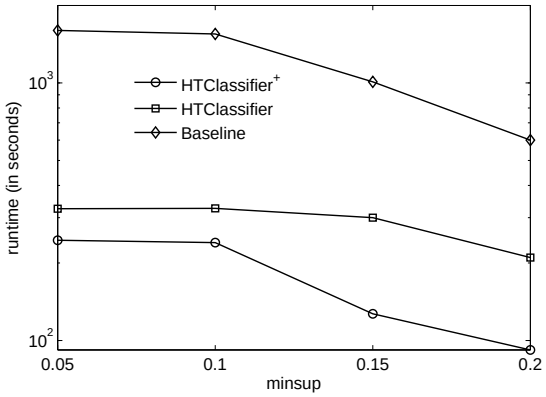
(b) Stulong Dataset

Figure 5.2. Effect of varying max_conf

We first vary the max_conf from 0.65 to 0.80. The default $minsup$ value is 5%. Figure 5.2 presents the runtime results for both the Hepatitis and Stulong datasets. We observe that as



(a) Hepatitis Dataset



(b) Stulong Dataset

Figure 5.3. Effect of varying $minsup$

max_conf increases, the runtimes of all 3 algorithms increase. This is because there are fewer rules when max_conf is high but the length of each rule is longer, and mining longer patterns requires more time. In particular, HTClassifier is at least one order of magnitude faster than the baseline algorithm for both datasets. The optimization strategies in HTClassifier+ are effective in reducing the runtime further.

Next, we vary $minsup$ from 5% to 20%, and set max_conf to 0.80. Figure 5.3 presents the results. Increasing $minsup$ reduces the number of patterns generated and improves the runtime.

This behavior is observed in both datasets. The optimizations enable HTClassifier+ to outperform HTClassifier with an average 78% reduction in search space and an average 41% improvement in runtime on the Hepatitis dataset. Further, HTClassifier+ reduces 68% search space and 40% runtime on the Stulong dataset.

5.4.2 Effectiveness Experiments

In this section, we demonstrate the usefulness of the discovered HT patterns by comparing the accuracies of the classifiers built using HT patterns to the classifiers built using patterns from single kind of data.

The HT patterns used to built the classifiers are obtained by running HTClassifier+ with $minsup$ = 5%, min_conf = 20% and max_conf = 80%. We build three classifiers based on HT patterns discovered. First, we create a working database DB' using the discovered HT patterns. Suppose the discovered HT patterns from a dataset DB are $\{\alpha_1, \alpha_2, ... ,\alpha_N\}$. Each instance in DB' has N attributes, one for each HT pattern, and a class attribute. The value of the $i-th$ attribute of an instance in DB' is set to 1 if the corresponding instance in DB supports α_i for $i \in [1, N]$. The class label of this instance in DB' is set to the class label of the corresponding instance in DB. We note that $|DB'| = |DB|$. Having obtained DB', we can now apply any classification method such as decision tree (J48), support vector machine (SVM), and boosting strategy (ADTree) to learn the rules from it. We denote HT_{J48} for the classifier built using $J48$ on DB'; HT_{Boost} for the classifier built using the boosted decision tree $ADTree$ on DB'; HT_{SVM} for the classifier built using SVM on DB'.

We also construct 3 classifiers using only patterns from categorical data (Cat_{J48}, Cat_{Boost}, Cat_{SVM}); 3 classifiers using only patterns from numerical data (Num_{J48}, Num_{Boost}, Num_{SVM}),

3 classifiers using patterns from categorical and numerical data ($\{Cat, Num\}_{J48}$, $\{Cat, Num\}_{Boost}$, $\{Cat, Num\}_{SVM}$); and 2 classifiers using patterns from interval data ($Interval_{seq}$, $Interval_{temporal}$). We use the default parameters for each classifier unless otherwise specified.

Three measures are used to evaluate the performances of the classifiers. Let P, N denote the number of positive and negative instances respectively; TP, TN denote the number of true positive and true negative respectively; FP, FN denote the number of false positive and false negative respectively; and TPR, FPR denote the rate of true positive and rate of false positive respectively. TPR determines a classifier performance on classifying positive instances correctly among all positive samples available during the test, $\frac{TP}{P}$. FPR defines how many incorrect positive results occur among all negative samples available during the test, $\frac{FP}{N}$. The three measures are:

- Accuracy = $\frac{TP+TN}{P+N}$
- Weighted F-measure (F-measure) = $\frac{2*TP}{2*TP+FN+FP}$
- Area under the curve $(AUC) = 0.5 + \frac{TPR-FPR}{2}$.

Figure 5.4 presents the 10-fold cross validation results of the various classifiers on the Hepatitis and Stulong datasets respectively. We observe that the classifiers built based on HT patterns significantly outperform the other classifiers in terms of accuracy, F-measure, and AUC. This indicates that HT patterns have higher discriminative power compared to patterns involving only single kind of data.

Next, we investigate the effects of max_conf and $minsup$ on the classification performance of HT patterns (see Figure 5.5). We note that varying $minsup$ from 2% to 20% does not change the classification performance of RuleClassifier and MetaClassifier. This is because the all essential HT rules found from the Hepatitis Dataset has support values $>$ 20%. On the other

Classifier	Accuracy(%)	F-measure	AUC
HT_{J48}	99.56	0.99	0.99
HT_{Boost}	99.56	0.99	0.99
HT_{SVM}	97.82	0.97	0.97
Cat_{J48}	59.13	0.50	0.56
Cat_{Boost}	59.78	0.49	0.59
Cat_{SVM}	60.86	0.53	0.53
Num_{J48}	60.65	0.52	0.60
Num_{Boost}	61.73	0.55	0.62
Num_{SVM}	65.00	0.57	0.57
$\{Cat, Num\}_{J48}$	60.00	0.51	0.59
$\{Cat, Num\}_{Boost}$	62.17	0.54	0.63
$\{Cat, Num\}_{SVM}$	60.44	0.53	0.53
$Interval_{seq}$	76.00	0.73	0.72
$Interval_{temporal}$	60.00	0.37	0.5

(a) Hepatitis Dataset

Classifier	Accuracy(%)	F-measure	AUC
HT_{J48}	96.25	0.96	0.97
HT_{Boost}	94.61	0.94	0.95
HT_{SVM}	95.67	0.95	0.95
Cat_{J48}	63.00	0.63	0.54
Cat_{Boost}	66.50	0.58	0.51
Cat_{SVM}	67.60	0.56	0.50
Num_{J48}	61.18	0.61	0.57
Num_{Boost}	63.57	0.61	0.57
Num_{SVM}	67.25	0.54	0.51
$\{Cat, Num\}_{J48}$	63.00	0.63	0.54
$\{Cat, Num\}_{Boost}$	66.50	0.58	0.51
$\{Cat, Num\}_{SVM}$	67.60	0.56	0.50
$Interval_{seq}$	56.42	0.49	0.49
$Interval_{temporal}$	67.21	0.42	0.51

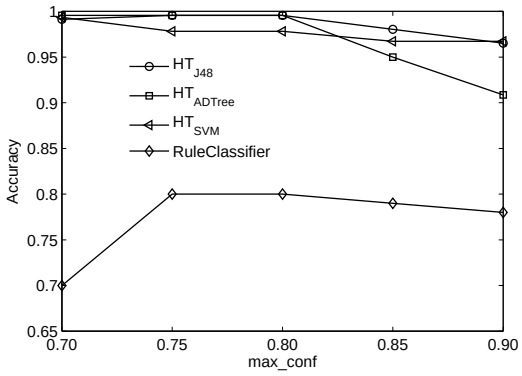
(b) Stulong Dataset

Figure 5.4. Evaluation results of the classifiers

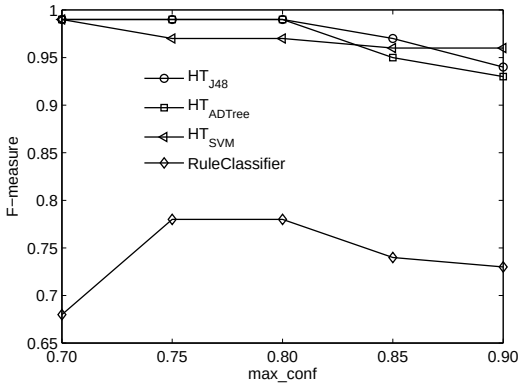
hand, the accuracy and F-measure of RuleClassifier and MetaClassifier peaks for lower value of max_conf. As max_conf increase, these values dips slightly. This could be because the high confidence HT patterns tend to overfit the data or we could not find any HT patterns that satisfy the $minsup$ and max_conf requirements. Note that, the reported classification performance is based on 10 fold cross-validation.

Figure 5.6 shows the effects of max_conf and $minsup$ on the classification performance of HT patterns on the Stulong Dataset. We observe that all four evaluation measures has a good value when max_conf = 0.80. As we increase max_conf, the performance drops. We observe that when max_conf is very high, we could not find good set of essential HT patterns that satisfy the max_conf requirement. In other word, there exists few training examples which are not covered at all. Thus, the designed classifier behaves like ZeroR for those uncovered examples. Similarly, when we vary $minsup$ beyond 0.1, the number of essential HT patterns decreases significantly leading to a sharp drop in the classification performance of HTMiner except Accuracy. Note that, Stulong dataset is a biased dataset and reported classification performance is based on 10 fold cross-validation.

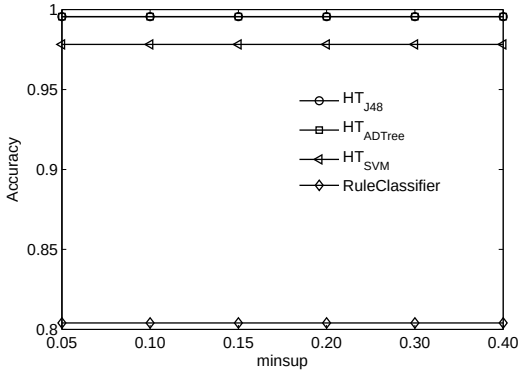
Figure 5.7 lists some of the essential HT patterns discovered from Hepatitis dataset along with their supports and associated class labels. Here, CL, CHE, UN and LAP are the names of some laboratory tests. The laboratory test results are indicated by 'N' signifying 'Normal level', 'L' signifying 'Low level', and 'H' signifying 'High level'. For example, CL_N denotes that the result of the laboratory test CL is normal. A quick survey of medical journals and publications reveal that some of the HT patterns have been noted by researchers previously, thus confirming the validity of these patterns.



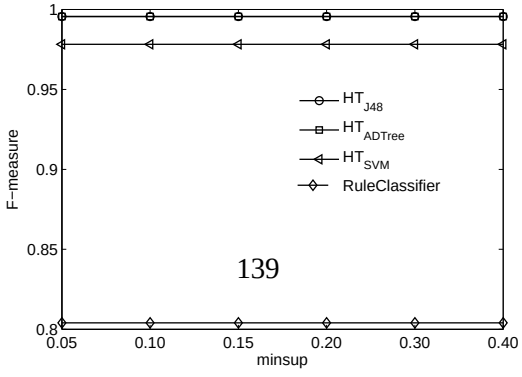
(a) Effect of max_conf on accuracy



(b) Effect of max_conf on F-measure

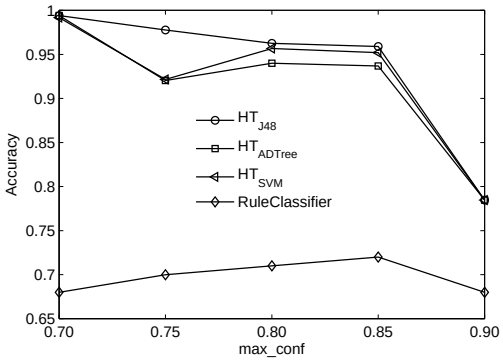


(c) Effect of $minsup$ on accuracy

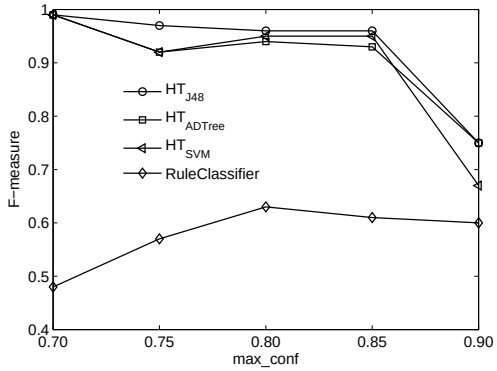


(d) Effect of $minsup$ on F-measure

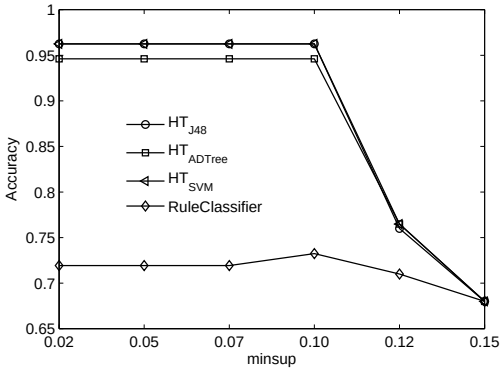
Figure 5.5. Evaluation of classifiers on Hepatitis dataset



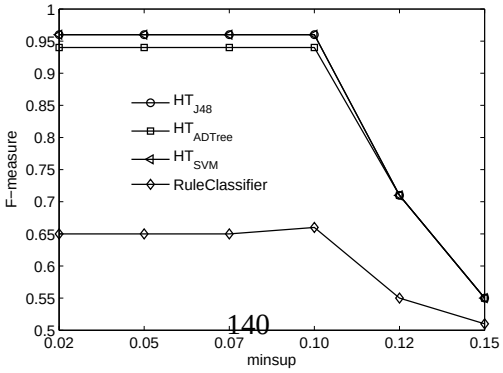
(a) Effect of max_conf on accuracy



(b) Effect of max_conf on F-measure



(c) Effect of $minsup$ on accuracy



(d) Effect of $minsup$ on F-measure

Figure 5.6. Evaluation of classifiers on Stulong dataset

	HT patterns	Class	Support	Confidence
1	α = [{CL_N,CHE_N}, $\emptyset$, {UN_N $\overset{Equal[0,0,0,0,0]}{\longrightarrow}$ LAP_N}, $\emptyset$]	HepC	32.60%	90%
2	α = [{ZTT_N,CL_N}, $\emptyset$, {T-BIL_N $\overset{Equal[0,0,0,0,0]}{\longrightarrow}$ UN_N}, $\emptyset$]	HepB	21.7%	90%
3	α = [{ALP_L,GPT_H},$\emptyset$, $\emptyset$, $\emptyset$]	HepB	6%	90%
4	α = [{Male}, $\emptyset$, {D-BIL_H}, $\emptyset$]	HepB	4%	90%

Figure 5.7. Example of HT patterns discovered from Hepatitis dataset

Figure 5.8 lists some interesting HT patterns discovered from the Stulong dataset. The first pattern indicates that a patient who complains chest pain over a period of time[1] has a higher risk of getting CVD. The third pattern states that a patient exercises regularly without shortness of breath has a lower risk of cardiovascular disease even if he has a history of high blood cholesterol and is a smoker. This confirms the usefulness of physical activity[2].

	HT patterns	Class	Support	Confidence
1	α = [$\emptyset$,{ChestPain},$\emptyset$,$\emptyset$]	CVD	14.60%	93%
2	α = [{$\emptyset$},{(MaxCigarettes,[1,3]),(AvgCigarettes,[0.60,1.1])}, {{NoChestPain $\overset{Finish\_By[0,1,0,0,0]}{\longrightarrow}$ NoDyspnea}},{$\emptyset$}]	non-CVD	8%	81%
3	α = [{HighBloodCholesterol}, $\emptyset$, {ModestActivity $\overset{Finish\_By[0,1,0,0,0]}{\longrightarrow}$ NoChangeInSmoking $\overset{Finish\_By[0,2,0,0,0]}{\longrightarrow}$ NoDyspnea},$\emptyset$]	non-CVD	7%	80%

Figure 5.8. Example HT patterns discovered from Stulong dataset

Exploration Order

In HTClassifier, we explore the search space in the following order: categorical data, followed by numerical data, then interval data, and finally time series data (i.e., C $\rightarrow$ N $\rightarrow$ I $\rightarrow$ T). This set of experiments investigates if changing the order of exploration will result in a significant difference in the performances of the resulting classifiers. Figure 5.9 presents the results of 10 fold

[1] http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1502238/
[2] http://www.americanheart.org/presenter.jhtml?identifier=4726

cross-validation for the classifiers using the default parameters. We observe that the exploration order has minimal effect on the classifier performances.

	Order	Acc(%)	F-measure	AUC
HT_{J48}	C → N → I → T	99.56	0.99	0.99
	I → C → N → T	98.91	0.98	0.99
	N → C → I → T	97.39	0.97	0.98
HT_{Boost}	C → N → I → T	99.56	0.99	0.99
	I → C → N → T	92.60	0.92	0.97
	N → C → I → T	94.78	0.94	0.98
HT_{SVM}	C → N → I → T	97.82	0.97	0.97
	I → C → N → T	97.17	0.97	0.97
	N → C → I → T	95.65	0.95	0.94

(a) Hepatitis Dataset

	Order	Acc(%)	F-measure	AUC
HT_{J48}	C → N → I → T	96.25	0.96	0.97
	I → C → N → T	96.37	0.96	0.95
	N → C → I → T	95.90	0.96	0.95
HT_{Boost}	C → N → I → T	94.61	0.94	0.95
	I → C → N → T	94.15	0.94	0.91
	N → C → I → T	93.68	0.93	0.91
HT_{SVM}	C → N → I → T	95.67	0.95	0.95
	I → C → N → T	95.67	0.95	0.95
	N → C → I → T	95.20	0.95	0.95

(b) Stulong Dataset

Figure 5.9. Effect of exploration order on classifiers' performances.

5.5 Summary

In this chapter, we have addressed the problem of mining patterns from datasets with different kinds of data. We introduced the notion of an heterogenous pattern to capture the association among patterns discovered from different kinds of data. Our contributions are:

1. We present an algorithm named HTMiner to mine complete set of frequent patterns from datasets with multiple kinds of data. HTMiner employs four optimizations and prefix based indexing techniques to enhance the efficiency.

2. We also present an algorithm named HTClassifier to mine essential set of discriminative patterns from dataset with multiple kinds of data for classification. HTClassifier utilizes sequential coverage based approach to mine a set of discriminative patterns for classification. We build a classifier using discovered patterns.

3. We extend the existing prefix based pattern growth approach for mining interval data. Further, the extended algorithm can mine temporal patterns and time lag temporal patterns together.

4. We validate the algorithm on real world datasets. Our experimental results show that the proposed approach is efficient. We also discover previously unknown patterns from the hepatitis and stulong datasets. Further, we show the usefulness of HT patterns discovered from both datasets by constructing a classifier and improving the classification accuracy.

In future, we would like to devise an incremental HTClassifier as data under consideration are updated regularly.

Chapter 6

Conclusions and Future Work

In this thesis, we investigated issues related to mining a specific class of datasets where the record contains observation from categorical, numerical, interval and time series data. For efficient realizations, we argued that algorithmic optimizations are essential to obtain efficiency that is commensurate with the data complexity. We designed novel algorithms and heuristics for the following three problems - mining temporal patterns from interval data; mining lag patterns from time series data; and developing a unified algorithm for analyzing dataset with multiple kinds of data. In addition to devising new algorithms, we also showed the usefulness of discovered patterns by applying them in real world applications.

In terms of novel pattern mining algorithms, we made the following contributions:

- We examined the problem of mining relationships among interval-based events. We augmented existing hierarchical representation with additional count information to make the representation lossless. Based on this new representation, we developed an Apriori-based IEMiner algorithm to mine frequent temporal patterns from interval-based events. We designed an efficient support counting procedure. The performance of IEMiner is further improved by

employing an event list blacklisting strategy and a prefix counting strategy. Experiments on synthetic data sets and real world datasets demonstrated the efficiency and scalability of our proposed approach. Beyond this, we designed the first interval-based classifier, IEClassifier to improve the predictive accuracy of closely related classes. Experiment results on the Hepatitis and Stulong datasets showed that IEClassifier outperforms traditional classifiers such as C4.5, CBA, and SVM.

- Next, we mined lag patterns from time series data. Our proposed approach extracted the repeated subsequences of various lengths from each time series entity. We used orderline concept and subsequence matching property to fulfil this requirement. Next, we described algorithm LPMiner that utilized inverted lists and various optimization strategies to improve runtime efficiency. Our experimental results demonstrated that the proposed approach is scalable and meaningful patterns can be discovered from stock dataset, stulong dataset and hepatitis dataset.

- We motivated mining patterns from datasets with multiple kinds of data. We introduced the notion of an heterogenous pattern to capture the association among patterns discovered from different kinds of data. We described two efficient algorithms named HTMiner and HTClassifier. Our algorithms employ a prefix based indexing method with optimization strategies to achieve good scalability with a reduction of search space compared to non-optimized algorithms. Experiments on two real world datasets indicated that the classifier built based on heterogenous patterns easily outperforms classifiers that was built using only patterns involving single kind of data.

6.1 Future Research Directions

There are several promising directions in which one can extend the work presented in this thesis. Applications that produce and process more complex data types such as graph data and image data are ubiquitous. Examples include social networks, retina dataset, bioinformatics, communication networks, world wide web, to name a few. A promising direction to extend our framework is to incorporate more complex data types. Further, in such applications, data is incremental. Hence, incremental learning methods can be designed to enhance the efficiency.

Pattern mining in spatio-temporal dataset assumes that spatial events are instantaneous and discover frequent sequential pattern such as $\{$low temperature $\rightarrow$ high percepitation $\rightarrow \cdots \}$ in near by region. However, many real world spatial events have duration. For example, forest fire in west Indonesia's jungle lasts 10 days. With the help of event duration, we can discover well known Allen's temporal relation among nearby spatial events and further leverage the discovered temporal relations in identifying cause and effect relation. Recently, many spatio-temporal database designers realize the usefulness of event duration in explaining many real world phenomena and thus have extended their frameworks to record event duration. With the growing demand of such dataset, we need a data mining approach which considers duration of spatial event and discovers temporal pattern. Note that, this dataset is dynamic hence we also need an algorithm which can works an incremental fashion.

Bibliography

[1] Ecml knowledge discovery challenege. *PKDD*, 2004.

[2] Casas smart home project - http://ailab.eecs.wsu.edu/casas/, 2009.

[3] Informs data mining contest, 2009.

[4] Drug safety: Observational medical outcomes partnership challenge, 2010.

[5] Y. Sheikh A. Hakeem and M. Shah. A hierarchical event representation for the analysis of videos. *AAAI*, 2004.

[6] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *SIGMOD*, pages 94–105, 1998.

[7] R. Agrawal, T. Imielinski, and A. Swami. Database mining: A performance perspective. *Special Issue on Learning and Discovery in Knowledge-Based Databases*, pages 914–925, 1993.

[8] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. *SIGMOD*, pages 207–216, 1993.

[9] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. *VLDB*, pages 487–499, 1994.

[10] R. Agrawal and R. Srikant. Mining sequential patterns. *ICDE*, pages 3–14, 1995.

[11] R. Agrawal and R. Srikant. Mining sequential patterns: Generalizations and performance improvements. *EDBT*, pages 3–17, 1996.

[12] James F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11), 1983.

[13] C. Antunes and A. L. Oliveira. Discovery of temporal patterns - learning rules about the qualitative behaviour of time series. *European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 192–203, 2001.

[14] C. Antunes and A. L. Oliveira. Generalization of pattern-growth methods for sequential pattern mining with gap constraints. *Lecture Notes in Computer Science*, pages 239–251, 2003.

[15] M. Atallah. Detection of sets of episodes in event sequences: Algorithms, analysis and experiments. *Thesis*, 2003.

[16] J. Augusto. Temporal reasoning for decision support in medicine, 2005.

[17] Yonatan Aumann and Yehuda Lindell. A statistical theory for quantitative association rules. *Intelligent Information Systems*, pages 261–270, 1999.

[18] C. bettini, X. Wang, and S. Jajodia. Testing complex temporal relationships involving multiple granularity and its application to data mining. *PODS*, 1996.

[19] C. Borgelt. An implementation of the fp-growth algorithm. *Proceedings of the 1st international workshop on open source data mining: frequent pattern mining implementations*, pages 1 – 5, 2005.

[20] S. Brin, R. Motwani, and J. D. Ullman. Dynamic itemset counting and implication rules. *http://infolab.stanford.edu/ sergey/dic.html*.

[21] G. Chen, X. Ma, D. Yang, S. Tang, and M. Shuai. A bipartite graph framework for summarizing high dimensional binary categorical and numerical data. *SSDBM*, pages 580–597, 2009.

[22] J. Chen. Data differentiation and parameter analysis of a chronic hepatitis b database with an artificial neuromolecular system. *Biosystems*, pages 23–36, 2000.

[23] M.-S. Chen, J. Han, , and P.S. Yu. Data mining: An overview from a database perspective. *TKDE*, pages 866–883, 1996.

[24] H. Cheng, X. Yan, J. Han, and P. S. Yu. Direct discriminative pattern mining for effective classification. *ICDE*, pages 169–178, 2008.

[25] B. Chiu, E. Keogh, and S. Lonardi. Probabilistic discovery of time series motifs. *SIGKDD*, 2003.

[26] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, September 2001.

[27] C.L. Isbell D. Minnen, I. Essa and T. Starner. Detecting subdimensional motifs: An efficient algorithm for generalized multivariate pattern discovery. *ICDM*, 2007.

[28] G. Das, K. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. *SIGKDD*, pages 16–22, 1998.

[29] Luc Dehaspe. Ruse-warmr: Rule selection for classifier induction in multi-relational data-set. *ICTAI*, pages 10–16, 2008.

[30] A. Denton. Density-based clustering of time series subsequences. *Mining Temporal and Sequential Data*, 2004.

[31] T. G. Dietterich and R. S. Michalski. Discovering patterns in sequences of events. *Artificial Intelligence*, pages 187–232, 1985.

[32] K. Eamonn and L. Jessica. Clustering of time-series subsequences is meaningless : implications for previous and future research. *Knowledge and information systems*, 8(2):154–177, 2005.

[33] G Baselli et al. Causal relationship between heart rate and arterial blood pressure variability signals. *Medical and Biological Engineering and Computing*, 26(4):374–378, 1987.

[34] C. Faloutsos. Fastmap: a fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. *SIGMOD*, pages 163–174, 1995.

[35] W. Fan, K. Zhang, H. Cheng, J. Gao, X. Yan, J. Han, P. S. Yu, and O. Verscheure. Direct mining of discriminative and essential graphical and itemset features via model-based search tree. *SIGKDD*, 2008.

[36] M. N. Garofalakis, R. Rastogi, and K. Shim. Spirit: Sequential pattern mining with regular expression constraints. *VLDBJ*, pages 223–234, 1999.

[37] F. Giannotti, M. Nanni, D. Pedreschi, and F. Pinelli. Mining sequences with temporal annotations. *SAC*, pages 593–597, 2006.

[38] D. Goldin, R. Mardales, and G. Nagy. In search of meaning for time series subsequence clustering: Matching algorithms based on a new distance measure. *CIKM*, 2006.

[39] G. Grahne and J. Zhu. Fast algorithms for frequent itemset mining using fp-trees. *TKDE*, pages 1347–1362, 2005.

[40] AJF Griffiths, SR Wessler, RC Lewontin, WM Gelbart, DT Suzuki, and JH Miller. Introduction to genetic analysis. *W.H. Freeman and Co*, 8th, 2005.

[41] H. Grosskreutz and S. Ruping. On subgroup discovery in numerical domains. *ECML*, 2009.

[42] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. Freespan: Frequent pattern-projected sequential pattern mining. *SIGKDD*, pages 355–359, 2000.

[43] Jiawei Han. How can data mining help bio-data analysis. *SIGKDD*, 2002.

[44] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, September 2000.

[45] F. Hoopner. Discovery of temporal patterns. learning rules about the qualitative behaviour of time series. *PKDD*, pages 192–203, 2001.

[46] P.S. Kam and A.W.C. Fu. Discovering temporal patterns for interval-based events. *Int. Conf. Data Warehousing and Knowledge Discovery*, pages 317–326, 2000.

[47] E. Keogh. Time series data mining tutorial. *Person Communication*, 2006.

[48] N. Lavrac, P. Flach, B. Kavsek, and L. Todorovski. Adapting classification rule induction to subgroup discovery. *ICDM*, pages 266–273, 2002.

[49] J. Lee, Y. Lee, B Hun H, and K Ryu. Discovering temporal relation rules from interval data. *Lecture Notes in Computer Science*, pages 57–66, 2002.

[50] J. Lin, E. Keogh, S. Lonardi, and P. Patel. Finding motifs in time series. *Proceedings of the Second Workshop on Temporal Data Mining*, 2002.

[51] B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. *SIGKDD*, pages 80–86, 1998.

[52] W.P.D. LOGAN. Mortality in the london fog incident. *Lancet*, i:336–338, 2005.

[53] W. Loh, S. Kim, and K. Whang. A subsequence matching algorithm that supports normalization transform in time-series databases. *DMKD*, pages 5–28, 2004.

[54] H. Mannila, H. Toivonen, and I. Verkamo. Discovery of frequent episodes in event sequences. *ICDE*, pages 210–215, 1995.

[55] T. Matsuda, H. Motoda, T. Yoshida, and T. Washio. Beam-wise graph-based induction: Mining patterns from structured data by. *Discovery Science*, pages 422–429, 2002.

[56] F. Michael, D. Phillip, and R. Deb. Mining temporal patterns of movement for video content classification pages. *Proceedings of the eighth ACM Int. Workshop on Multimedia Information Retrieval*, pages 183 – 192, 2006.

[57] D. Minnen, C.L. Isbell, I. Essa, and T. Starner. Discovering multivariate motifs using subsequence density estimation and greedy mixture learning. *AAAI*, 2007.

[58] D. Minnen, T. Starner, I. Essa, and C. Isbell. Activity discovery: Sparse motifs from multivariate time series. *Snowbird Learning Workshop*, 2006.

[59] D. Minnen, T. Starner, I. Essa, and C. Isbell. Discovering characteristic actions from on-body sensor data. *Int. Symp. on Wearable Computing (ISWC)*, 2006.

[60] D. Minnen, T. Starner, I. Essa, and C. Isbell. Improving activity discovery with automatic neighborhood estimation. *Int. Joint Conf. on Artificial Intelligence*, 2007.

[61] T. M. Mitchell. Machine learning. *McGraw Hill*, 1997.

[62] F. Moerchen. Algorithms for time series knowledge mining. *SIGKDD*, pages 668 – 673, 2006.

[63] A. Mueen, E. Keogh, and N. Bigdely-Shamlo. A disk-aware algorithm for time series motif discovery. *ICDM*, 2009.

[64] A. Mueen, E. Keogh, Q. Zhu, and S. Cash. Exact discovery of time series motifs. *SDM*, 2009.

[65] E. Muller, I. Assent, and T. Seidi. Hsm: Heterogeneous subspace mining in high dimensional data. *SSDBM*, pages 497–516, 2009.

[66] R Nevatia, T. Zhao, and S. Hongeng. Hierarchical language-based representation of events in video streams. *IEEE Workshop on Event Mining*, 2003.

[67] S. Nijssen, T. Guns, and L. D. Raedt. Correlated itemset mining in roc space: a constraint programming approach. *SIGKDD*, pages 647–656, 2009.

[68] T. Oates. Peruse: An unsupervised algorithm for finding recurring patterns in time series. *ICDM*, pages 330 – 337, 2002.

[69] S. Papadimitriou, J. Sun, and C. Faloutsos. Streaming pattern discovery in multiple time-series. *VLDB*, pages 697–708, 2005.

[70] S. Papadimitriou, J. Sun, and P.S.Yu. Local correlation tracking in time series. *ICDM*, pages 456 – 465, 2006.

[71] P. Papapetrou, G. Kollios, and S. Sclaroff. Fluent learning: elucidating the structure of episodes. *In Proc. IDA*, pages 268–277, 2001.

[72] P. Papapetrou, G. Kollios, and S. Sclaroff. Discovering frequent arrangements of temporal intervals. *ICDM*, 2005.

[73] D. Patel, Wynne Hsu, and Lee Mong Li. Mining multiple kinds of data for effective classification. *Submitted to SIGKDD for Review*, 2011.

[74] D. Patel, Wynne Hsu, Lee Mong Li, and Srinivasan Parthasarathy. Lag patterns in time series databases. *DEXA*, 2010.

[75] Dhaval Patel, Wynne Hsu, and Lee Mong Li. Mining relationships among interval-based events for classification. *SIGMOD*, 2008.

[76] J. Pei, J. Han, and R. Mao. Closet: An efficient algorithm for mining frequent closed itemsets. *SDM*, pages 21–30, 2000.

[77] J. Pei, J. Han, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. Prefixspan: mining sequential patterns efficiently by prefix-projected pattern growth. *ICDE*, pages 215–224, 2001.

[78] K. A. Peker. Subsequence time series (sts) clustering techniques for meaningful pattern discovery. *Integration of Knowledge Intensive Multi-Agent Systems*, pages 360–365, 2005.

[79] H. Pinto, J. Han, J. Pei, K. Wang, Q. Chen, and U. Dayal. Multi-dimensional sequential pattern mining. *CIKM*, pages 81–88, 2001.

[80] M. Plantevit, Y.W. Choong, A. Laurent, D. Laurent, and M. Teisseire. M^2sp: Mining sequential patterns among several dimensions. *PKDD*, pages 205–216, 2005.

[81] J. F. Roddick, K. Hornsby, and M. Spiliopoulou. Temporal, spatial and spatio-temporal data mining and knowledge discovery research bibliography. *http://kdm.first.flinders.edu.au/IDM/STDMBib.html*.

[82] R. Ronkainen. Attribute similarity and event sequence similarity in data mining. *Thesis, University of Helsinki*, 1998.

[83] L. Sabra, J. Anne, and P. Andrew. The xs and y of immune responses to viral vaccines. *The Latent Infectious Disesases*, pages 338–349, 2010.

[84] Y Sakurai, S Papadimitriou, and C Faloutsos. Braid: Stream mining through group lag correlations. *SIGMOD*, 2005.

[85] P. Shenoy, J. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah. Viper: A vertical approach to mining association rules. *SIGMOD*, 2000.

[86] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. *SIGMOD*, pages 1–12, 1996.

[87] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. *EDBT*, 1996.

[88] A. Vahdatpour, N. Amini, and M. Sarrafzadeh. Toward unsupervised activity discovery using multi-dimensional motif detection in time series. *IJCAI*, 2009.

[89] J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences, 2003.

[90] Geoffrey Webb. Discovering associations with numeric variables. *SIGKDD*, pages 383–388, 2001.

[91] I. H. Witten and E. Frank. Data mining: Practical machine learning tools and techniques. *Morgan Kaufmann*, 2005.

[92] Di Wu, G. Fung, J. Xu Yu, and Z. Liu. Mining multiple time series co-movements. *APWeb*, pages 572–583, 2008.

[93] S. Wu and Y. Chen. Mining nonambiguous temporal patterns for interval-based events. *TKDE*, pages 742–758, 2007.

[94] X. Yan, H. Cheng, J. Han, and P. S. Yu. Mining significant graph patterns by leap search. *SIGMOD*, pages 433–444, 2008.

[95] X. Yan, J. Han, and R. Afshar. Clospan: Mining closed sequential patterns in large datasets. *PKDD*, pages 166–177, 2003.

[96] G. Yang. The complexity of mining maximal frequent itemsets and maximal frequent patterns. *SIGKDD*, 2004.

[97] D. Yankov, E. Keogh, J. Medina, B. Chiu, and V. Zordan. Detecting time series motifs under uniform scaling. *SIGKDD*, pages 844–853, 2007.

[98] X. Yin, J. Han, J. Yang, and P. S. Yu. Efficient classification across multiple database relations: A crossmine approach. *TKDE*, 18(6):770–783, 2006.

[99] T. Yoshiki, I. Kazuhisa, and U. Kuniaki. Discovery of time-series motif from multi-dimensional data based on mdl principle. *Machine Learning*, 58(2-3):269–300, 2005.

[100] M. J. Zaki and C. Hsiao. Charm: An efficient algorithm for closed itemset mining. *SIGKDD*, pages 457–473, 2002.

[101] M.J. Zaki. Spade: An efficient algorithm for mining frequent sequences. *Machine Learning J.*, pages 31–60, 2001.

[102] Y. Zhu and D. Shasha. Statstream: Statistical monitoring of thousands of data streams in real time. *VLDB*, 2002.