

DISCOVERING TEMPORAL PATTERNS FOR INTERVAL-BASED EVENTS

KAM, Po-shan

A Thesis Submitted in Partial Fulfillment
of the Requirements for the Degree of
Master of Philosophy
in
Computer Science and Engineering

©The Chinese University of Hong Kong
June 2000

The Chinese University of Hong Kong holds the copyright of this thesis. Any person(s) intending to use a part or whole of the materials in the thesis in a proposed publication must seek copyright release from the Dean of the Graduate School.



論文題目：區間的時態模式
作者：甘寶珊
學院：香港中文大學
學部：計算機科學與工程
修讀學位：哲學碩士

摘要

在許多日常的應用程式中，我們收集到不同時間性的數據，例如：銷售記錄，股票買賣，病錄表，地理學和天文學的科學數據。這些資料包含了時間的概念，記錄了事件的發生時間。從這些順序的數據中我們能更清楚了解事件發生的型態、趨勢，從而找出各樣物件的關係。假若如有大量的數據顯示某些事件的發展模式，我們便能找出事件之間發生的次序和特性。在這篇論文裡，我們探討有關從時序性的數據中，尋找事件與事件之間的關係和型態，例如："事件 A 出現於事件 B 發生的期間"。

在現有的方中，大部分的運算都將時間的發生看作點狀 (Point-based)，並作先後排列。因此，只能用很簡單的型式表達事件的發展，缺乏能力表達較為複雜的時態關係，例如：在...期間，互相重疊等。加上在日常應用的程式中，不難發現點狀 (Point-based) 和區間 (Interval-based) 的時序性數據並存於資料中。我們有必要發展一些新的方式，去支援區間數據的運算。在這裡我們介紹了兩種包含區間時態的數據模式，分別為 AppSeq 和 LinkSeq。這兩種模式不但簡單而且對於描述事件的行為很有用。我們並發展了幾個方法，利用不同的數據結構 (Data structure)，去協助加快運算的過程。在這些方法中，我們運用了一組被廣泛應用於描繪區間關係的型態，用作描述兩件事件之間的關係。透過對一連串人造和真實的數據集作研究來驗證我們所提出的方案能有效地應用於大量數據中。

Discovering Temporal Patterns for Interval-based Events

submitted by

KAM, Po-shan

for the degree of Master of Philosophy
at the Chinese University of Hong Kong

Abstract

Sequence of data can be collected in many applications. Examples range from sales records, stock exchange, patient records, to scientific databases in geophysics and astronomy. Such databases incorporate the concept of time which describes when an event starts and ends as historical records. The temporal nature of data provides us with a better understanding of trend or pattern over time so as to find any correlation between events. An important and interesting characteristic of event sequences can be found if the collection of events occur in a certain pattern. In this thesis, we are interested in discovering temporal relations between events which satisfy certain timing constraints, e.g. “event A appears *during* the period when event B occurs”.

Existing algorithms for mining temporal pattern treats data as chronological orders of event sequences and most of them support point-based events. Therefore, the physical ordering of events would be quite simple and there have been limited expressive power in specifying temporal relations such as *during*, *overlaps*, etc. Moreover, it is likely that both point and interval-based data may exist and co-exist within many application domains. To address these problems, we introduce two kinds of patterns, namely *AppSeq* and *LinkSeq*, which accommodate temporal interval data. Both patterns are simple and useful to describe the behavior of the events. We develop several methods for finding such interesting patterns in which we use different data structures to facilitate efficient mining process. In these methods, we propose to use a generalized taxonomy of temporal relationships which is highly expressive to describe the basic relationships between two events. A quantitative performance study was conducted through experiments on synthetic and real datasets and the results show the efficiencies of the proposed methods for large databases.

Acknowledgments

This thesis is dedicated to my parents and my brother who support me to pursue a master degree, and for their continuous support throughout my studies.

I wish to express my deepest gratitude to my supervisor, Prof. Ada Fu, for her continuing support and guidance. I thank her for her encouragement, enlightening discussions and advice which have been an invaluable resources for me. I would also like to thank Prof. Pheng-ann Heng and Prof. Man-hong Wong for their comments and insights on this work.

I am thankful to my friends, who inspired me to have my further study as a M.Phil student. They are Chun-chun Tong, Hiu-lam Lam, Chi-man Lam, Chi-fung Wong, Lai-kuen Mak, Hon-wai Fung and Sun-on Cheung. In the course of my graduate studies, I have benefited from the interactions with many warm people around me. I would especially like to thank Yuk-ming Chan, Tsui-ying Law, Kwong-wai Chen, Ka-po Ma, Wai-chiu Wong, Wai-ching Wong, Chun-hing Cai, Tze-kin Lao, Yin-ling Cheung and Po-man Wan for their endless encouragement and enthusiasm, and for making these years truly enjoyable.

Finally, I want to thank God for His love and grace to me. I believe He works with me throughout the years. I thank my brothers and sisters in church for their prayers, continuous encouragement, love and the patience they showed me.

Contents

Abstract	i
Acknowledgements	ii
1 Introduction	1
1.1 Data Mining	1
1.2 Temporal Data Management	2
1.3 Temporal reasoning and temporal semantics	3
1.4 Temporal Data Mining	5
1.5 Motivation	6
1.6 Approach	7
1.6.1 Focus and Objectives	8
1.6.2 Experimental Setup	8
1.7 Outline and contributions	9
2 Relevant Work	10
2.1 Data Mining	10
2.1.1 Association Rules	13
2.1.2 Classification	15
2.1.3 Clustering	16
2.2 Sequential Pattern	17
2.2.1 Frequent Patterns	18
2.2.2 Interesting Patterns	20
2.2.3 Granularity	21
2.3 Temporal Database	21
2.4 Temporal Reasoning	23
2.4.1 Natural Language Expression	24
2.4.2 Temporal Logic Approach	25

2.5	Temporal Data Mining	25
2.5.1	Framework	25
2.5.2	Temporal Association Rules	26
2.5.3	Attribute-Oriented Induction	27
2.5.4	Time Series Analysis	27
3	Discovering Temporal Patterns for interval-based events	29
3.1	Temporal Database	29
3.2	Allen's Taxonomy of Temporal Relationships	31
3.3	Mining Temporal Pattern, AppSeq and LinkSeq	33
3.3.1	A1 and A2 temporal pattern	33
3.3.2	Second Temporal Pattern, LinkSeq	34
3.4	Overview of the Framework	35
3.4.1	Mining Temporal Pattern I, AppSeq	36
3.4.2	Mining Temporal Pattern II, LinkSeq	36
3.5	Summary	37
4	Mining Temporal Pattern I, AppSeq	38
4.1	Problem Statement	38
4.2	Mining A1 Temporal Patterns	40
4.2.1	Candidate Generation	43
4.2.2	Large k -Items Generation	46
4.3	Mining A2 Temporal Patterns	48
4.3.1	Candidate Generation:	49
4.3.2	Generating Large $2k$ -Items:	51
4.4	Modified AppOne and AppTwo	51
4.5	Performance Study	53
4.5.1	Experimental Setup	53
4.5.2	Experimental Results	54
4.5.3	Medical Data	58
4.6	Summary	60
5	Mining Temporal Pattern II, LinkSeq	62
5.1	Problem Statement	62
5.2	First Method for Mining LinkSeq, LinkApp	63
5.3	Second Method for Mining LinkSeq, LinkTwo	64
5.4	Alternative Method for Mining LinkSeq, LinkTree	65

5.4.1	Sequence Tree: Design	65
5.4.2	Construction of seq-tree	69
5.4.3	Mining LinkSeq using seq-tree	76
5.5	Performance Study	82
5.6	Discussions	85
5.7	Summary	85
6	Conclusion and Future Work	87
6.1	Conclusion	87
6.2	Future Work	88
	Bibliography	97

List of Figures

2.1	KDD process	11
2.2	A sample FP-tree	15
2.3	Sample decision tree and Naive Bayesian network	16
2.4	An event structure	21
2.5	Versions of employee objects, and a time index	23
3.1	The thirteen possible relationships between two intervals X and Y	31
3.2	A sequence of interval-based events	33
3.3	Different temporal pattern representation	33
3.4	Differences between LinkSeq and AppSeq	35
4.1	Composition of two item-list, L_2 and L_1	43
4.2	The main algorithm	44
4.3	The candidate generation algorithm	45
4.4	Support counting for the candidates	47
4.5	The candidate generation algorithm of AppTwo	50
4.6	Forming L_2 without C_2	53
4.7	Distribution of temporal relations between events	54
4.8	Variation on minimum support	55
4.9	Execution time for each pass	56
4.10	Number of large items generated in each pass	56
4.11	Variation on window size	57
4.12	Scale-up: Number of sequences	58
4.13	Scale-up: Number of events per sequence	59
5.1	A seq-tree constructed for a sequence	66
5.2	Identical Pattern if same main-bh and sub-bh	67
5.3	Splitting of node C for different sub-bh	68
5.4	The seq-tree constructed using the given example	72

5.5	Construction of sequence tree, seq-tree	73
5.6	Building main branch	74
5.7	Adding subsidiary branches	75
5.8	Mining frequent temporal pattern from seq-tree	81
5.9	Variation on minimum support	83
5.10	Scale-up: Number of sequences	84
5.11	Scale-up: Number of events per sequence	85

List of Tables

4.1	Transform the database as <i>seq-list</i> and <i>item-list</i>	41
4.2	Partial Large 1-item list	43
4.3	The 3-candidates	44
4.4	Partial large 2-item list	45
4.5	Partial large 4-item list formed by AppTwo	49
4.6	Parameters	53
4.7	Number of AppSeq with different <i>min_sup</i>	57
4.8	Mining AppSeq in a scoliosis database	61
5.1	Large 1-items	69
5.2	Transform the database as sequence form	70
5.3	Mining LinkSeq by traversing <i>seq_tree</i>	79
5.4	Number of LinkSeq obtained with different <i>min_sup</i>	83

Chapter 1

Introduction

The research in this thesis grows out from the development of data mining in temporal databases. With the extension of mining temporal pattern which accommodates interval data, interesting and useful temporal information can be found. In this first chapter, we give a brief introduction of data mining and temporal database which provide the basic principles of the work being investigated. Related work on temporal reasoning and temporal data mining is also discussed. Then we address the motivation and approach of the work undertaken.

1.1 Data Mining

In the last decade, *data mining* or *knowledge discovery in databases* (KDD), has emerged as a significant field of research [59, 61, 38, 49, 27]. This emergence has been motivated by the rapid development of data warehousing which intelligent analysis of data is required. Moreover, with the advances in technologies such as the widespread use of bar codes in supermarket goods, monitoring devices in hospitals, sensors on-board orbiting satellites for scientific and geophysical science investigation, enormous amount of data which most of them are of high dimension are collected. The explosive growth of data makes it infeasible to analyze them manually and thus leads to a promising field of study, called data mining. Data mining is defined as *the nontrivial extraction of implicit, previously unknown, and potentially useful information from data* [27].

Data mining is developed from the confluence of research in machine learning, statistics and database systems [61, 38, 27]. The fundamental goals of data mining are prediction and description. From the existing variables in the databases, we predict unknown or future values of interest by the knowledge obtained. Also by finding

frequent patterns which well describe the behavior of the data, we can have a better understanding about the system for further analysis. On the other hand, the size and complexity of data is generally large, efficient and scalable algorithms are needed. The discovered knowledge can be applied to commercial industry for making better marketing strategy, decision making system or expert system for medical diagnosis and geographical information system or scientific data as a tool for analyzing data. A survey of current data mining issues is given in Chapter 2.

Although there have been many studies of data mining in transaction database, there are other applicative databases such as temporal databases, spatial databases, object-oriented databases and multi-media databases, etc., which requires specific data mining techniques to facilitate efficient and effective knowledge discovery on particular kinds of data. Advances in the research work on temporal data structures, temporal reasoning, indexing and query languages for temporal databases provides new challenges to the study of temporal data mining. However, there are still not many data mining techniques are extended to accommodate the specific properties of the temporal data. The focus of this thesis is on the methods of temporal data mining which extracts the temporal information stored in temporal databases.

1.2 Temporal Data Management

Temporal databases incorporate the concept of time to maintain past, present and future data [78, 83, 40]. They store time-varying information. As most database applications are temporal in nature, e.g., financial applications such as portfolio management, accounting, and banking, record-keeping applications such as personnel, medical-record, and inventory management, and scientific applications such as weather monitoring, the study of temporal databases has been an active field of research in the past decade.

Generally, a temporal database supports three distinct types of time attributes which are *valid time*, *transaction time* and *user-defined time*. Valid time stored the time when an event takes place with start time and end time values. Transaction time is the time when the event is recorded in the database and user-defined time is an uninterpreted time domain. Since valid time describes the occurrence pattern of events stored in the database, it promises greater utility as a source of domain knowledge than transaction time. Hence, the discussion of temporal pattern in this thesis will be focused upon valid time in temporal database.

Each record stores the *start time* and *end time* during which the tuple is valid.

Data is collected in the form of event time sequences where each event lasts for a certain time interval. For instance, in hospital information systems which laboratory examinations or clinical records are stored for medical diagnosis of patients' behavior over a certain monitoring period. Records like "patient A had surgery from 10:00 to 13:00 on 14 June" are stored. The temporal nature of data provides us a better understanding of trend or pattern over time so as to find any valuable information. For example, patterns like "60% of patients who took medicine A and then took medicine B after an hour, got a fever the following day" can be found. The frequent temporal patterns exhibited by patients may identify some correlations between drugs for further diagnosis. Other temporal data such as telecommunication network, weathering and marketing data in which by analyzing sequences of time-stamped data, we can have a better understanding of the data which changes over time. Knowledge discovery in temporal databases thus catches the attention of researchers [69, 16].

Moreover, recent research on temporal databases has made important contributions in characterizing the semantics of temporal information and in providing expressive and efficient means to model, store, and query temporal data [12, 40]. Different models of database management for efficient storage and access of temporal data are proposed. Optimization of query processing and indexing techniques are also under active investigation. For instance, an extended SQL standard, TSQL2, has been developed for temporal databases [72]. Within the TSQL2 standard, time is widely represented by intervals defined between start time and end time points. Queries with interval as primitive are adopted. In other words, both point-based and interval data are supported. We believe these significant investigations for the development of temporal databases, such as temporal data structure [10], temporal algebraic operators [81], query processing [80, 12], indexing [23], etc., have paved the way for the study of temporal data mining. As existing mining techniques cannot be applied to temporal databases to handle the temporal interval data directly, new algorithms of knowledge extraction is needed to capture the temporal semantics.

1.3 Temporal reasoning and temporal semantics

On the other hand, considerable research effort has been directed to the temporal aspects of information systems. One of the work is temporal reasoning which involves the issues of time modeling and the representation of temporal relationships based on the underlying temporal domain. Basically, there are two primitive notion of temporal data, *time point* and *time interval*, which temporal reasoning systems based

on. Time points are assumed to be linear and an ordering relation is defined. While intervals are expressed in a pair of start time and end time points (I^-, I^+) , with $I^- < I^+$ such that the ordering relations are expressed in terms of relations between their endpoints. Unlike time points, intervals can have complex inter-relations, for example, an interval may overlaps, meets or before another interval.

Other different representations of the semantics of temporal information are proposed also. They have different measures of ordering and metric relationships which helps to express and reason about time in many application domains. One of the formalisms for time modeling is temporal logics. They follow the syntax and semantics from modal logic to represent temporally definite statements by means of temporal operators [30, 53]. The development of various forms of temporal logic has played a part in data mining research, particularly in temporal pattern matching and sequence mining. For instance, first order temporal logic is used as a way to represent temporal patterns [56, 11]. Sequence such as (analyst A recommends “*buy*” for a stock until analyst B recommends “*sell*”) can be expressed as “`Analyst_Report (analyst, stock, recommendate) \wedge \neg o Analyst_Report (analyst, stock, recommendate)`” where $\{\wedge, o\}$ are operators meaning “And” and “Next” respectively. However, temporal logics are computationally intractable and have an expressive power that exceeds the requirement of most temporal databases. Therefore, a number of formalisms that weakens the temporal logic expressiveness have been developed.

Some of them define an algebra of temporal relationships according to a classical point of view. One of the most commonly used interval-based formalism is Allen’s interval algebra [9]. It models the relationship between any two intervals as a subset of a set of thirteen basic relations, including *before*, *meets*, *overlaps*, *starts*, *during*, and *finishes*, together with *their inverses*, plus the relation *equal*. Binary operations of intersection and composition are defined on the set of relationships. Allen’s interval algebra as well as first order temporal logic are most widely used in corporation with knowledge discovery process.

We incorporate temporal reasoning as a mean for the representation of temporal knowledge in our framework. A strong emphasis is placed on the complexity of the mining result which should be easily read and comprehended. Hence, the selection of an appropriate set of temporal predicates is fundamental to provide useful temporal reasoning. In our problem, Allen’s thirteen temporal relationships [9] is adopted to describe the basic binary temporal predicates and details would be covered in Chapter 3.

1.4 Temporal Data Mining

Temporal data constitutes a large portion of data collected in daily operations. In general, temporal data can be loosely defined as any data that contains temporal information. Examples include financial database for stock price index, telecommunications and medical databases. Searching for similar patterns in a temporal database is useful in many applications as we can discover and predict the risk, causality, and trend associated with a specific pattern. The accommodation of time into mining techniques provides a window into the temporal arrangement of events and thus an ability to suggest cause and effect or trends in rule sets. Temporal data mining is thus an important extension as it has the capability to infer causal and temporal proximity relationships that non-temporal data mining is not able to do.

The time component we captured helps in analyzing the changes of the data over time of the system. We may find any causal relationships from the ordering of occurrences of events such as the first condition which is followed by the second one is identified as cause and effect relationship other than association if no knowledge of time is known. Likewise, the time component may assist in identifying the validity of rules like “Hiking Boots \Rightarrow Outerwear”, Years \cdot Months(3:5) during [Years(1990), Years(1995)]” [17]. It reveals that every spring time from 1990 to 1995, the customers who buy hiking boots also buy outerwear. Such a rule may not be valid before 1990 or after 1995. We observe that by adding the temporal semantics to the rule set, more accurate and clear information is obtained. In addition, by discovering the change in knowledge obtained in the underlying data, it is possible to know how quickly the domain is likely to change which helps in better marketing strategies [62]. For example, by identifying frequently or unexpected occurring patterns over event sequences such as stocks with similar price movement, customer’s purchasing patterns over seasons as well as rare events happened for fraud detection, we gain more information from the sequences of records.

In general, a set of historical data is collected in the form of event time sequences. Current temporal data mining techniques can be broadly classified into two categories: categorical and numerical data analysis. The former one focuses on the discovery of causal relationships among temporally-oriented events. Most of the events concerned are point-based categorical events where only the time when the transaction takes place is recorded like sales records, telecommunication network alarms, etc. Some of the categorical data are interval-based events that the valid time are supported by the system such as patient database, scientific databases in geophysics and astronomy

areas, etc. The ordering of data is a valuable source of information which can direct future operations. Numerical data analysis concerns the discovery of similar patterns within the same time sequence or among different time sequences [25, 5, 20]. Numerical values of the sequences are taken into consideration as a comparison for trend discovery and prediction and it is known as time series analysis [21, 33]. Different shapes of the changes of data over time are analyzed [6].

Previous work for knowledge discovery in temporal data mainly work on sequential pattern [1, 8, 51, 50]. Although potential knowledge can be extracted, these techniques only treat data as series in chronological order. They consider the ordering of a string of events and thus mainly support point-based events. Hence most of these algorithms ignore time intervals which the data is stamped with. The physical ordering of events would be quite simple and there have been limited expressive power in specifying temporal relations such as *during*, *overlaps*, etc. To address these problems, we introduce two kinds of patterns namely *AppSeq* and *LinkSeq*, which accommodate temporal interval data. and discuss in details in Chapter 3.

1.5 Motivation

The motivation behind of this research is to extend the work of temporal data mining which examines interval-based data stored in temporal databases. In view of the emerging needs of temporal data mining and the problem of addressing temporal interval data, we aim to find common sequences that accommodates the temporal semantics of interval data. We introduce the problem of mining temporal patterns for interval-based events with the following observations.

1. There are emerging need for the development of temporal databases that capture the temporal nature of data stored in many applications [78]. Studying information stored in temporal databases lead us to have a better understanding of the evolving business. Moreover, the rapid development on research on temporal databases, different models of temporal data for storage and query processing are suggested. This favors the work for developing temporal data mining techniques.
2. Mining sequence data for interval-based events is important as besides finding association between temporal data, the ordering or relation between events provide us some insight into causal relationships. Besides before/after relation, other descriptions of temporal relations can be specified for interval data, which

helps in understanding the general trend of the sequence data. However, most of the existing algorithms work on point-based data only. Simple ordering of events is considered where series or parallel ordering of events are taken place. Hence they cannot be applied to temporal database directly where valid time is supported. There is a need to extend the existing work of mining sequential pattern to accommodate interval-based events.

3. Contrary to the case of time points, relationship among time intervals can be described in different ways. As interval can form different structures other than only before/after relation so that we can have a better understanding of how the events interact with each other. A generalized taxonomy of temporal relationships which is simple and highly expressive is needed to express the complex relations between intervals. Besides, as more complex relations involved, there are possibly vast number of temporal relationships can be found for a single sequence of events, and many of them may be too complicated and not useful to the user. We restrict our interest to simple and meaningful type of temporal pattern but yet the pattern found is highly expressive to reflect the complex relations among events. This motivates us to explore two interesting temporal patterns, AppSeq and LinkSeq in our work.
4. We believe that both patterns are useful to describe the temporal behavior among events. For example, given clinical records storing time varying attributes we can find AppSeq and LinkSeq among data; given stock market data, we can analyse the changes of data among different intervals of time; etc. Hence, besides specifying the before/after relationship, by considering timing-interval restrictions, we obtain other interesting knowledge from temporal data.

1.6 Approach

In our framework, we mainly focus on temporal databases which store interval-based events and discover any interesting temporal relations among them, in order to find any correlation between events. For example, in the medical field, patterns like “60% of patients who contract disease A got the disease *during* the time where disease B is also contracted” can be found. The frequent temporal patterns of diseases exhibited by patients may identify some correlations between diseases that can provide invaluable information for diagnosis.

1.6.1 Focus and Objectives

As mentioned before, there is tremendous forms of temporal patterns can be derived from a sequence of interval-based events. However, we notice that the complexity of the results increases as we introduce more complicated combinations, which may not be a desirable feature. Also, the computation time required would be increased and may not be feasible for mining purpose. Hence we do not consider complex temporal patterns. We here limit our focus on two temporal patterns which both reveal the temporal behavior among events. We believe that the temporal relations give some insight into causal relationships. As such, when a few events have happened, together they may become the cause of a following event. Both temporal patterns gives us a modeling of this idea. We introduce the patterns in Chapter 3 and methods for mining these patterns are given in Chapter 4 and 5.

1.6.2 Experimental Setup

To evaluate the performance of the proposed methods over a large range of data, we conducted several experiments on UltraSparc 5/270 workstation with 520MB of main memory. All methods are written in C.

First, we consider a set of synthetic data in an application domain of a medical database. The database stores person-id, disease the person contracted and the corresponding duration of time. For each person, we record a sequence of clinical records stating different diseases contracted. Each such sequence is potentially a maximal large sequence. An example of such a sequence might be “person A contracts disease X and during the treatment of disease X, disease Y is contracted”. The number of events per sequence of each person is chosen from Poisson distribution around a mean and a few person may have many clinical records where each record refers to an event in a sequence. We will use the synthetic database for mining both patterns.

Secondly, we work on a real data set which contains clinical records of Scoliosis patients. Scoliosis refers to spinal deformation. The database stores a list of measurements on the patients, such as the number of curves, the curve locations, degree of curvature, curve directions, etc. It also records patents’ personal information such as date of birth, family history, the class of Scoliosis contracted and the treatment. Sequences of records of about 900 patients are stored. Short sequences containing one or two records are obtained and for some patients, longer sequences are found. By examining the changes in values of some temporal attributes in the view of a sequence, we may discover any temporal knowledge stored in the database. We aim to

find any interesting patterns that occur frequently and hence discover any correlation between other non-temporal attributes.

1.7 Outline and contributions

This thesis focuses on the work of temporal data mining. A framework for mining temporal patterns is suggested. We introduce the notion of temporal representation which is capable of expressing the relationships between interval-based events. Two interesting types of temporal patterns are considered. We believe our findings can lead to useful systems in mining temporal patterns involving events that have a duration.

We start our discussion by surveying current research in data mining in Chapter 2. The aim of this survey is to provide a better understanding of the nature of knowledge discovery. Moreover, we present issues of temporal data and semantics in the context of temporal database management. This gives a background knowledge of the complexities of integrating temporal semantics into data mining techniques which would be discussed later. We also present the state of the art of temporal data mining by showing different directions of current work and identify related research challenges in the area.

Chapter 3 provides an introduction of the mining problem. In a sequence of interval-based events, we are interested in finding temporal relationship between events along the time-line. In particular, we focus on two kinds of temporal patterns, *AppSeq* and *LinkSeq* which both are simple and easy to understand. A general framework of the two temporal patterns are described.

Chapter 4 investigates in greater depth of the first temporal pattern, *AppSeq*. We would further introduce the variations of *AppSeq* as *A1* and *A2* patterns, where *A2* pattern being the variates of *A1* pattern. We introduce the notion of these temporal patterns and describe the methods for finding them. Experiments on both synthetic data and real data set are presented.

Chapter 5 considers the problem with another point of view such that another kind of temporal pattern *LinkSeq* is suggested. We introduce several methods for mining the second temporal pattern by means of different data structures to facilitate efficient support counting process. A performance study was conducted through experiments on synthetic data sets and the results show the efficiency of the proposed methods.

Finally, we give a conclusion and talk about our future work in Chapter 6

Chapter 2

Relevant Work

In this chapter, we briefly introduce the current issues of research on data mining, their directions and challenges involved. We then study the recent work on temporal database management and temporal reasoning in Section 2.3 and 2.4 respectively. They provide some background knowledge about the problem we discussed later. In Section 2.5, we investigate some related work of various temporal data mining techniques together with their new challenges faced. One of the problems of accommodating interval data is addressed in this thesis.

2.1 Data Mining

In recent years, the rapid growth in the size of databases has led to an increased interest in the automatic extraction of knowledge from data [59, 61, 49, 27]. The term **data mining**, or **knowledge discovery in databases (KDD)**, has been adopted to the general concept of seeking knowledge from data held in more or less structured databases [59, 27]. Strictly, KDD can be viewed as the overall process of extracting useful and interesting information from databases. This process includes the selection and preparation of data, manipulation and analysis of the result obtained. Data mining thus can be considered as part of the KDD process. Figure 2.1 shows an overview of the KDD process. It mainly divides into the following phases:

Understanding the domain: As KDD is a discovery driven process, we need to have a solid understanding of the domain in order to select the right subsets of data, suitable classes of patterns, and good criteria for interestingness of the pattern concerned.

Cleaning data: With some missing values or invalid data by incorrect input,

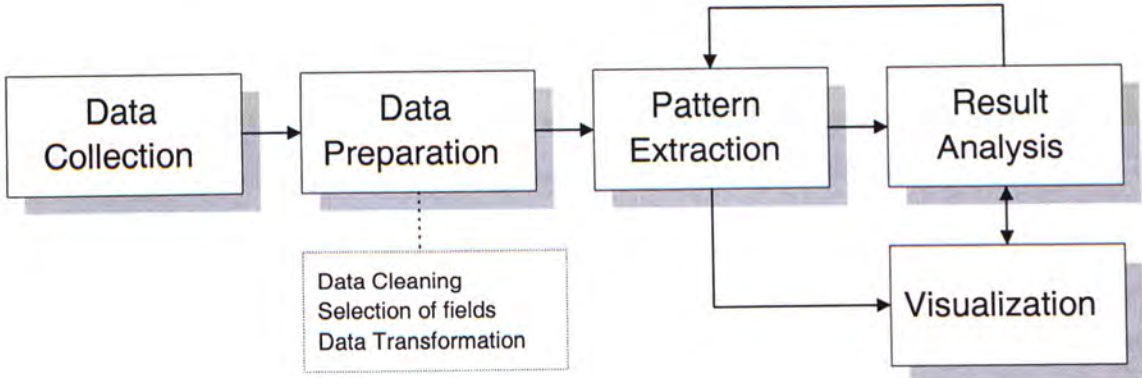


Figure 2.1: KDD process

different ways of exploiting useful data from databases are proposed. Selection of essential fields used in the mining process is also performed in this phase.

Discovering patterns or data mining: Scalable and effective mining algorithms are required for the extraction of interesting knowledge over a large set of data. Different strategies for scaling down the search space in the mining process are proposed.

Postprocessing of discovered patterns: Further analysis of the discovered patterns with expert knowledge is performed. Any evaluation of the result can feedback to the mining process to improve the quality of information obtained.

Presenting final result: A user-friendly interface is recommended to visualize the mining result for making the discovered patterns easier to understand. Interactive analysis of final result can be made also.

Data mining incorporates methods and tools of three areas: machine learning, statistics, and databases [49]. All three areas aim at locating interesting regularities, patterns, or concepts from empirical data, while data mining emphasizes on automatic knowledge discovery from huge data and the data can be corrupted by noise, errors or missing values. Moreover, there are many kinds of data and databases used in different applications which contain complex data types such as structured data hypertext, object-oriented databases, multimedia databases, spatial databases, temporal database [66], spatio-temporal databases [28] and transaction databases. Different techniques that facilitate efficient and effective extraction of information are needed. We summarize the issues and challenges of the development of data mining.

- **Efficiency and Effectiveness** As huge amount of data is being considered, scalability of the mining process becomes the main concern over the last decade. At the same time, effectiveness of the algorithms for finding useful patterns is also the key concerns for developing various mining methods.
- **Interestingness of knowledge obtained** The discovered knowledge should accurately portray the contents of the database and be useful for certain applications. Most of the measures of interestingness currently used are based on statistical measure of frequency which provides “best”, “optimal” or “most interesting” rules [43, 41]. We may interested in finding customer’s buying preference from sales records and thus we look for any frequently occurring pattern. However, in some cases, we may interested in finding rare events such as fraud detection in credit card payment, plan failures from plan execution traces [86], or exceptional rules in medical field for diagnosis [82, 73]. Different measures of interestingness are made.
- **Various types of data** With the development of various types of databases which involve complex data, it requires different mining techniques to cope with the specific intrinsic information embedded in the data. Example ranges from temporal data, spatial data and multimedia data to semi-structured web documents.
- **Robust to outliers** As mentioned before, most of the real data contains incomplete information or unexpected values regarded as noise, which requires careful handling to avoid any discrepancies of the mining result.
- **Interactive mining process** Data mining techniques are usually application dependent, hence any expert knowledge incorporated interactively in the mining process helps in producing useful results. Moreover, for any database, the amount of knowledge extracted may be far greater than that of the original data set. Therefore, a multi-stage of filters that reduce the query search space on the basis of source data, target pattern, statistics and significance of the mining result is needed. An interactive environment which provides a flexible way for users to determine the number of rules or pattern obtained in different instant is preferred.
- **Visualization of mining result** Visualization of data mining result supports interactive mining process at multiple abstraction levels [42, 70]. Also,

it helps to present the mining result to users in a more user-friendly way by means of a nice graphical interface.

Data mining techniques are usually used associated with decision support systems and knowledge base creation like data warehouses or expert systems where upon the newly discovered knowledge is used to improve system performance or provide better strategy. On the other hand, knowledge obtained can be used for detection of inconsistencies and integrity enforcement in some systems or semantic query optimization used in data warehouse.

Among various data mining techniques, four major data mining tasks are association, classification, clustering and sequence discovery. An excellent survey of different aspects of KDD was conducted in [27]. Here, we provide a brief introduction of each of these four areas as follows.

2.1.1 Association Rules

Mining association rules in transactional or relational databases has caught a lot of attention since [4]. Association rules typically find correlations between items in transaction data sets I that customers purchase several items in a single transaction. The relationship between items can be expressed as follows:

$$X_1 \wedge X_2 \wedge \cdots \wedge X_m \Rightarrow Y_1 \wedge Y_2 \wedge \cdots \wedge X_n$$

where $X \subset I$, $Y \subset I$, and $X \cap Y = \emptyset$. An association rule which derives from a database of transactions consists of a set of items bought by a customer in a single visit to a store. For example, an association rule can be “if a customer buys milks, he/she usually buys bread in the same transaction”. A set of rules are obtained for further interpretation. Usually we use support count and confidence value to measure the interestingness of the rules found. In other words, we focus on frequently occurring pattern. Applications include supermarket, inventory planning, attached mailing in direct marketing and promotional sales planning.

An *Apriori* level-wise method is proposed for mining the association rules. The algorithm starts from small data set to large ones using the anti-monotone *Apriori* heuristic: *if any length k pattern is not frequent in the database, its length $(k+1)$ super-pattern can never be frequent.* The mining process mainly divides into two phases, namely the candidate generation phase and test-and-bed phases [4]. Since mining association rules may require repeatedly scanning through a large transaction database to find different association patterns, the amount of computational cost

could be very high. Efficient algorithms for mining association rules using various data structure and pruning strategies for performance enhancement are developed [7, 57, 87, 71, 37]. For example, one of the recent work of finding frequent patterns for association rules is developed in [37]. A tree-like structure, FP-tree, is used for the mining process.

As [37] suggested, FP-tree is proposed to deal with frequent pattern especially for large data set and long patterns. It is an extended prefix-tree structure storing crucial, quantitative information about frequent patterns. The FP-tree is constructed in such a way that

- Only frequent length-1 items will have nodes in the tree. This ensure only potential frequent patterns which form from frequent length-1 items are considered.
- The set of frequent items of each transaction is stored in the tree in such a way that each item is added as a node in the tree. Tree nodes are arranged by descending order of frequency of length-1 items with higher frequency nodes being placed close to the root node. For each transaction, a list of nodes are inserted from the root and placed according to the frequency of the items.
- Multiple transactions sharing an identical frequent item set can be merged into one with the number of occurrences registered as count. It is easy to check whether two sets are identical if the frequent items in all of the transactions are sorted in descending order of frequency.
- Two transactions which share a common prefix, according to some sorted order of frequent items can merge the shared part using one prefix structure with the count registers properly. If the frequent items are sorted in descending order of frequency, more prefix strings can be shared and hence the size of the tree is reduced.
- Each transaction is mapped to one path in the tree, and the frequent itemset information in each transaction is stored in the tree. Since the frequent itemset in any transaction is always encoded in a corresponding path of the frequent pattern tree, traversing the tree ensures the completeness of the result.

The FP-tree of a transaction database is shown in Figure 2.2 with minimum support being 40%. By looking up each event in the header table, we find the corresponding paths containing the node and examine the prefix subpath of the node. With the use of the counter values stored in each node, the complete set of the frequent patterns can be generated. Since the structure of FP-tree helps in keeping all

TID	Items	(sorted) freq. items
1	f,a,c,d,m,p	f,c,a,m,p
2	a,b,c,f,l,m	f,c,a,b,m
3	b,f,h,j,o	f,b
4	b,c,k,s,p	c,b,p
5	a,f,c,e,p,m	f,c,a,m,p

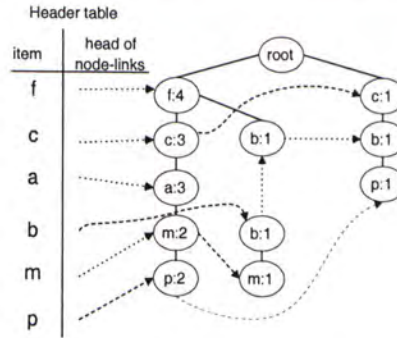


Figure 2.2: A sample FP-tree

the important information for support counting of frequent patterns. No further look up from the database is needed. In other words, it generates frequent pattern without the candidate generation phase as well as scanning of database for support counting. This greatly reduce the high cost for mining frequent pattern and experimental results also show the advantages of using this approach.

On the other hand, variations of association rules like mining sequential pattern, quantitative association rules from market basket data are suggested [8, 74, 75]. As sometimes the rules found are quite large in amount to digest and understand, further investigations about the interestingness of the rules discovered are carried out [43, 46, 41].

2.1.2 Classification

Classification is the process which finds the common properties among a set of objects in a database and classifies them into different classes, according to a classification model [3]. For a given set of records with its corresponding attributes, we categorize the records with similar attribute values within a group and describe the characteristics of each classes. Based on the history, a classification function is developed for identifying new candidates for predication. In credit analysis, the card issuing company will have customer records containing a number of descriptors. So for the customer with a known credit history, the customer’s record is tagged as “excellent”, “good”, “medium” or “poor”. One of the classification rules can be “customers with excellent credit history have a debt/equity ratio of less than 10%”. Such class descriptions

are then used to classify future incoming data of the databases. Applications range from target mailing, franchise location, credit approval, treatment-appropriateness determination to scientific data analysis such as SKICAT [26].

Data classification has been studied substantially in statistics, machine learning, neural networks, and expert systems, and is an important theme in data mining. Examples are decision trees [60] and bayesian network classifiers [29] shown in Figure 2.3. As classification aims at classifying new records to an appropriate class, accuracy for the classification result becomes the main concern of the problem. Various methods of improving the accuracies of the results like using entropy values of information theory, penalty of wrong classification, multiple classifiers are proposed. On the other hand, in recent years, extension to accommodate spatial objects for geographical studies is suggested [45].

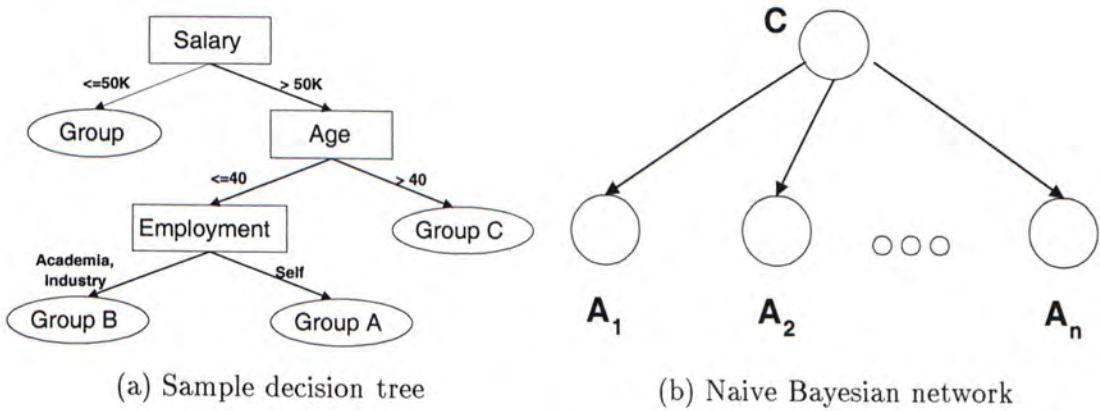


Figure 2.3: Sample decision tree and Naive Bayesian network

2.1.3 Clustering

Clustering segments an original database into different subsets or clusters. Its task is to identify clusters or densely populated regions, according to some distance measurement, from a large and multi-dimensional data set. Given a set of objects and a clustering criterion such as number of clusters required and distance measure of the objects, we group the objects into clusters such that the objects in a cluster are more similar to each other than to objects in different clusters. As it finds any interesting structure directly from the data without using any background knowledge, like concept hierarchies, clustering is useful for discovering groups and identifying interesting distributions in the underlying data.

Clustering techniques have been studied extensively in statistics [47]. Algorithms like Partitioning Around Medoids (PAM) or Clustering LARge Applications (CLARA)

are found to be inefficient from the computational complexity point of view. As for the efficiency concern, an algorithm called Clustering Large Applications based upon RANdomized Search (CLARANS), was developed [54]. Experimental results showed that CLARANS outperforms PAM and CLARA. As both PAM and CLARA find k representative points of k clusters from N objects by analyzing all possible pairs of objects, it is therefore computationally inefficient for large values of N and k . For clustering large sets of points, with the use of a *clustering feature tree* (CF) which summarizes the information about sub-clusters of points, the algorithm Balanced Iterative Reducing and Clustering (BIRCH) [88] is capable of finding good clusters with a single scan of the data.

On the other hand, the traditional clustering algorithms like k-mean, or k-medoids approaches suffer from the fact that a cluster is represented by just one point. These methods cannot deal with irregular shapes and sizes of different clusters in data. To address this problem, a method called Clustering Using Representatives (CURE) [32] using multiple representatives for a clusters is proposed. It chooses a set of well-scattered points to represent a cluster and employs a novel hierarchical clustering algorithm that adopts a middle ground between the centroid-based and the all-point extremes. Also, a density-based clustering method (DBSCAN) [24], which supports arbitrary shapes of clusters is suggested. Clusters such as concave clusters, clusters with noise and with significantly different diameters that are located close to one another can be found. In addition, the number of clusters can be unknown in advance and no input of such parameter is required.

The problem of high dimensionality can be tackled by using sub-space approach CLIQUE [2] for cluster analysis. As given a large set of multi-dimensional data points, the data space is usually not uniformly occupied by the data points. CLIQUE identifies the sparse and the crowded places by considering appropriate subspaces over the original dimensions, and hence maximize the similarity within a cluster and maximize the difference between other groups. This allows records with missing values to be used for clustering with more accurate results than replacing missing values taken from a distribution.

2.2 Sequential Pattern

Unlike the previous three classes of data mining problems where static data is captured, sequential pattern mining is a temporal knowledge extraction over a sequence of data where each data associates with an occurrence time. A linear ordering known

as serial ordering is obtained. The problem of mining sequential patterns can be stated as follows: Let $I = \{i_1, i_2, \dots, i_n\}$ be a set of *items*. An *itemset*, i is a subset of items and denoted as $(i_1 i_2 \dots i_k)$, where i_j is an item. A *sequence* is an ordered list of itemsets. A sequence α is denoted by $(\alpha_1 \mapsto \alpha_2 \mapsto \dots \mapsto \alpha_q)$, where the sequence *element* α_j is an itemset. A sequence $\alpha = (\alpha_1 \mapsto \alpha_2 \mapsto \dots \mapsto \alpha_n)$ is called a *subsequence* of another sequence $\beta = (\beta_1 \mapsto \beta_2 \mapsto \dots \mapsto \beta_m)$ if there exist integers $1 \leq j_1 < j_2 < \dots < j_n \leq m$ such that $\alpha_1 \in \beta_{j_1}, \alpha_2 \in \beta_{j_2}, \dots, \alpha_n \in \beta_{j_n}$. A sequence with k itemsets ($k = \sum_j |\alpha_j|$) is called a *k-sequence*. We are interested in finding any k -sequence with support above a given threshold (minimum support), where support is the number of data sequences that contain the pattern.

As many application domain have time attribute, data various from web log [73], telecommunication network [50], to marketing sales transactions [8] are collected. By analyzing the sequence of data, we obtain a better understanding of the trend or the behavior of a system. For example, consider the sales database of a book store which records the books bought by each customer over a period of time. Interesting patterns like “70% of the people who buys Jane Austen’s *Pride and Prejudice* also buys *Emma* within a month.” Stores can use these patterns for promotions, shelf placement, etc. Or in a web access database at a particular site, the discovered patterns are the sequences of most frequently accessed pages at that site. This kind of information can be used to restructure the web-site, or to dynamically insert relevant links in web pages based on user access patterns.

2.2.1 Frequent Patterns

Since the introduction of mining frequent patterns in a sequence database in [8], many studies have contributed to the efficient mining of sequential patterns or other frequent patterns in time-related data [51, 50, 87, 85, 37, 58, 36]. The mechanism proposed in [8] relies on the Apriori heuristic first proposed in association mining [4]: *any super-pattern of a non-frequent pattern cannot be frequent*. Frequent sequences of length k are built from frequent sequences of length $k - 1$ by applying a self-join operation to the latter set and computing the support of the resulting sequence. Finally, non-maximal frequent sequences are removed from the result. Based on this heuristic, a generalized definition of sequential patterns that include time constraints, sliding time window, and user-defined taxonomy is proposed and a generalized sequential pattern mining algorithm, GSP is developed [75].

On the other hand, research on sequence mining has been oriented towards the

discovery of *episodes* that occur frequently within sequences [51, 50]. An episode is formally a conjunction of events which includes serial and parallel ordering of events. The results of sequence mining are episode rules of the form $P[V] \Rightarrow Q[W]$ where V, W are time intervals. We may discover any causal relations among events. An extension work of finding frequent episodes [51, 50], which uses temporal logic as a formalism for expressing temporal patterns defined over categorical data is suggested [56]. It discovers frequent patterns which satisfy certain temporal logic expressions. Temporal logic programming is suggested as a mechanism for the discovery of frequent patterns expressible in temporal logic. First-order temporal logic, FOTL, is used to express patterns such as “Hold(Stock) until Bearish_Market_Sentiment”, where Hold is a temporal predicate. Temporal operators such as *since*, *until*, *next* are considered, which cannot be expressed in terms of episodes.

As Apriori employs a bottom-up search that enumerates every single frequent itemset, the exponential complexity is fundamentally inefficient for discovering long patterns or large data set with low minimum support. New algorithm, Sequential Pattern Discovery using Equivalence classes (SPADE) for fast discovery of sequential pattern is proposed [85]. By using a *vertical id-list* database format, where each sequence associate a list of objects in which it occurs, along with the time-stamps, all frequent sequences can be enumerated via simple id-list intersections. A lattice-theoretic approach to decompose the original search space into smaller pieces (sub-lattices) which can be processed independently is suggested. All sequences can be found by three scanning of the database. Other data structure like WAP-tree [58], which is used for mining sequential pattern, is a variation of FP-tree [37] as shown in Figure 2.2. WAP-tree is proposed for mining web logs data which point-based events are considered. By storing the critical information for access pattern mining, without the generation of large candidate set, WAP-tree facilitates efficient mining of access patterns. Besides, other technique like projection databases with the use of frequent item matrix [36] is introduced for mining sequential pattern. Its general idea is to use frequent items to recursively project sequence databases into a set of smaller projected databases and grow subsequence fragments in each projected database. Experimental results show that by reducing the high cost of candidate generation and test, these methods outperform Apriori-based GSP method.

2.2.2 Interesting Patterns

As conventional mining techniques provide users with limited mechanism (based on minimum support) for specifying patterns of interest, the “unfocused” approach suffers from two major drawbacks: (1) disproportionate computational cost for selective users, (2) overwhelming volume of potentially useless results. To tackle these problems, a simple and natural syntax of Regular Expressions (REs) is introduced to be used as a flexible constraint specification tool that enables user-controlled focus incorporated into the pattern mining process [31]. The RE constraints help in pruning the search space of patterns during computation as well as directing useful results which return to the users. A family of algorithms, namely Sequential Pattern mining with Regular expression constraints (SPIRIT), is proposed for mining frequent sequential patterns that satisfy user-specified RE constraints. The suggested algorithms address the problems by narrowing down the target search space. Similar idea of imposing selection constraints on user-specified pattern is suggested [34]. A language for specifying episodes of interest is used.

Besides using frequencies as a measure of interestingness of a sequential pattern, a notion of using minimum description length principle as a mean to evaluate mining result is proposed [14]. Based on the number of bits in which a sequence can be encoded under an appropriate coding scheme, sequences with large code length are interpreted as potentially surprising patterns and these patterns are considered to be interesting. Likewise, instead of finding frequently occurring patterns, [82, 11, 86] suggest another measure of interestingness to capture rare events. It is extremely useful for error discovery as errors are supposed to be rare events and if we restrict the search space to frequent sequences, these exceptional sequences would be easily rejected and impossible to be distinguished from any trivial sequences. By pruning out any predictive and redundant patterns that are out of interest, PlanMine which proposed to deal with plan failure prediction [86] and Timeweaver which applied to predict telecommunication equipment failures [82], reduce the size of the returned rule set significantly. A probabilistic approach of unexpectedness is adopted in [11]. A pattern P is deemed interesting if the ratio of the actual number of occurrences of P exceeds the expected number of occurrences of P by some user defined threshold. It is assumed that each event in the sequence occurs with some probability and certain conditional distribution exist between neighboring events. Based on this, an expected number of occurrences of a certain pattern in a sequence can be computed. Unexpected patterns like “((vtrace NEXT lseek) NEXT lseek)” is found from a

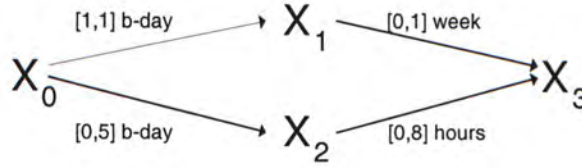


Figure 2.4: An event structure

series of operating system calls made by a sendmail program.

2.2.3 Granularity

The discovery of temporal patterns or relationships that involve multiple granularities is addressed in [13]. It is stressed that events occurring in the same day, or happening within k weeks from a specific day may capture our attention. With the use of an event structure which is a set of temporal constraints on a set of variables representing events, we target for patterns of events that match the even structure. Consider the event structure depicted in Figure 2.4, variables such as X_0 , X_1 , X_2 and X_3 can be assigned as *IBM-rise*, *IBM-earnings-report*, *HP-rise*, and *IBM-fall*, respectively. This complex event type describes the scenario that the IBM earnings were reported one buisness day after the IBM stock rose, and in the same or the next week the IBM stock fell; while the HP stock rose within five business days after the same rise of the IBM stock and within eight hours before the same fall of the IBM stock. To facilitate this pattern matching process, the notion of a *timed finite automaton with granularities* (TAG) is introduced. A TAG is essentially a standard finite automaton with the modification that a set of clocks is associated with the automaton and each transition is conditioned not only by an input symbol, but also by the values of the associated clocks and the clocks of an automaton may be running in different granularities.

2.3 Temporal Database

Temporal database records time-varying information. As most applications are temporal in nature, e.g. financial applications, inventory management, scheduling applications and scientific applications, the development of temporal databases becomes a vibrant research topic over the last two decades [78, 77, 39, 40] and a number of bibliographies of research in the field have been published [52, 76, 44, 83]. All this work has made important contributions in characterizing the temporal semantics of

information and in providing expressive and efficient means to model, store and query about temporal data.

Temporal database maintenance requires the consideration of different time dimensions [78]. Two orthogonal time dimensions have been proposed: *valid time*, i.e. the period during which the information input is valid, and *transaction time*, i.e., the time of the record is stored. While valid time allows us to keep track of the history of the application domain throughout its evolution in time, transaction time allows us to maintain the history of the evolution of the database.

Recent research on temporal databases can be roughly categorized into three areas. The first area is the formulation of the semantics of time [12] which is closely related to research issues in knowledge representation. Issues various from theoretical point of view such as temporal logic [30] and infinite periodic time sequences to rather applied questions such as how to represent time values in minimal space [10] and how to utilize calendars [15]. Moreover, data types such as time points, time intervals and temporal elements (sets of intervals) as the representation of time are discussed [78]. Since the data explicitly stored in a temporal database are often associated with certain semantic assumptions, each assumption can be viewed as a way of deriving implicit information from explicitly stored data.

The second area concerns the physical implementation issues which focus on efficient access methods and data organization strategies in temporal database [23]. Conventional indexes have long been used to reduce the need to scan an entire relation to access a subset of its tuples, to support the selection algebraic operator and temporal joins to facilitate efficient temporal query processing. Many temporal indexing strategies are based on B^+ -trees, which index on the time point values [23, 22, 10, 68]. One of these indexing techniques is *time index* [23] which is shown in Figure 2.5. It is capable of retrieving versions of object that are valid during a specific time period. It is proposed to improve the performance of certain classes of temporal queries such as *when*, *during*. The valid time intervals of various object versions will overlap in arbitrary ways. Since one cannot define a total ordering on the interval values, special attention is paid in the selection of the set of linearly ordered indexing points on the time dimension.

The third category of the study on temporal database is the logical modeling of temporal data where most of them emphasize an extension of relational data model to capture temporal semantics and to support relational temporal query languages [79, 77]. These extended models generally augment relations of the snapshot data model with several time attributes which store the relevant timestamps such as valid

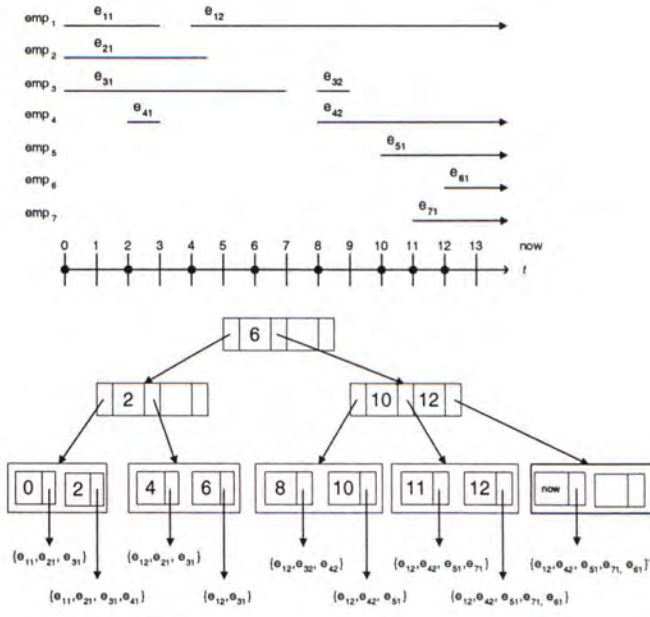


Figure 2.5: Versions of employee objects, and a time index

time and transaction time. New temporal operators are defined in these extended data models, based upon traditional relational algebraic operators, to allow users to query time attributes. The languages designed must support predicates on temporal values, multiple calendars, schema versioning, periodic data, point-based and interval-based semantics and have adequate expressive power and the ability to be efficiently implemented. One of the notable work on temporal querying language is TSQL2 [72]. TSQL2 is developed to consolidate approaches to temporal calculus-based query languages and is a comprehensive temporal extension to SQL2 in which time is represented by intervals.

2.4 Temporal Reasoning

Closely related to the study of formulation of the semantics of time in temporal database is the research work on temporal reasoning in Artificial Intelligence (AI) or Information System (IS) [53]. In temporal reasoning, there is a fundamental choice between whether time points or time intervals [9, 48] are the primitive objects to reason about action or time. One of the most influential theories of time, *interval algebra* (IA) has been introduced by James Allen [9]. It addresses the problem of representing temporal knowledge and performing temporal reasoning from the perspective of natural language understanding. The notion of *relations between pairs of intervals* is introduced. The bases of Allen’s approach consists of an *interval-based*

temporal logic, together with a computationally effective reasoning algorithm based on *constraint propagation*.

An interval I is represented as an ordered pair (I^-, I^+) of real numbers with $I^- < I^+$, denoting the left and right endpoints of the interval respectively. Relations between intervals are composed as disjunctions of *basic interval relations* which are known as *interval operators* that operate on two intervals (denoted by $int_1 = (s_1, e_1)$ and $int_2 = (s_2, e_2)$). A boolean value is returned as follows:

- int_1 overlaps $int_2 \equiv ((s_1 \leq s_2 \leq e_1) \vee (s_2 \leq s_1 \leq e_2))$
- int_1 during $int_2 \equiv ((s_1 \geq s_2) \wedge (e_1 \leq e_2))$
- int_1 meets $int_2 \equiv (e_1 = s_2)$
- $int_1 < int_2 \equiv (e_1 \leq s_2)$
- $int_1 \leq int_2 \equiv ((s_1 \leq s_2) \vee (e_1 \leq e_2))$

Based on these thirteen basic relations, we get $2^{13} = 8192$ possible relations between intervals in the full algebra. Hence reasoning with this algebra (that is, reasoning about implied interval relations or determining the consistency of a set of assertions), however has been shown to be NP-complete. Hence, *point algebra* (PA) is introduced which based on the notion of a time point in place of an interval. The basic relations of the PA that can hold between two points are $<$, $=$ and $>$. The relation between two points is a disjunction of the basic relationships which gives the set $\{<, \leq, >, \geq, =, \neq, \emptyset, ?\}$.

2.4.1 Natural Language Expression

Temporal representation and reasoning are necessary components of systems that consider events that occur in the real world. In appointment scheduling and time management, natural language expressions that refer to collections of intervals are used prevalently and routinely. Thus, an effective means of representing the intervals is essential. For example, some classes of expressions such as “the first day of every month” refer to a collection of intervals explicitly. On the other hand, expressions like “the U.S. Election Day: the first Tuesday after the first Monday in November” specify the intervals implicitly. Formula which represent collections of intervals are proposed [48]. For example, assume that time t_0 is Saturday, December 31,1994, midnight, the collection of *Thursdays* can be described by the formula:

$$\text{Thursdays} = \{ \langle \alpha; 1\text{day} \rangle \mid \alpha = 5\text{days} + t_0 \pmod{7\text{days}} \}$$

The foundation of the collection representation is a set of primitive collections called *calendars*. A calendar is a collection consisting of an infinite sequence of intervals that span the time-line. *Days*, *Months* and *Chinese-Calendar-Years* are instances of calendars.

2.4.2 Temporal Logic Approach

Besides using natural language expression for reasoning about time, a mathematical model which use temporal logics is proposed. The logics are based on the concept that instead of a predicate calculus statement being universally true or false, it may be true or false at different moments of time. Temporal quantifiers are used to augment the calculus. Temporal operators such as “since”, “until”, “next” are used.

2.5 Temporal Data Mining

Temporal data mining is *the non-trivial extraction of implicit, potentially useful and previously unknown knowledge from an implicit or explicit temporal content from large quantities of data* [67]. This accommodation of time into mining techniques provides a window into the temporal arrangement of events and thus an ability to suggest cause and effect.

2.5.1 Framework

It has been recognized recently that time dependent information is important in data mining [69]. Temporal patterns concerning temporal features of the rules such as associations should be investigated and discovered from temporal databases since they can provide accurate information about an evolving business domain. A prototype system architecture for mining temporal patterns is introduced [16]. A generic definition of temporal pattern is presented. Temporal pattern is defined as a triplet $\langle \text{Patt}, \text{PeriodicExp}, \text{IntervalExp} \rangle$, where *Patt* is a general pattern which may be a trend, a classification rule, an association rule, a causal relationship, etc., *PeriodicExp* is a periodic time expression and *IntervalExp* is a general interval expression. It takes absolute time as a measure of interest in the discovery of patterns. A temporal data mining language, Temporal Query and Mining Language (TQML) is suggested to be integrated into the framework. However, since the snapshots, i.e. *IntervalExp*, are derived without any knowledge of temporal patterns existing within the data, many interesting temporal patterns may be lost. On the other hand, the idea of adding

temporal semantics to existing data mining tasks is suggested in [62]. It introduces the integration of temporal reasoning into knowledge discovery process. Two conventional data mining algorithms are extended to handle temporal semantics which are attribute-oriented induction [63] and association rule [64].

2.5.2 Temporal Association Rules

Since the introduction of association rules [4], it has been extended in different ways to deal with quantitative and categorical data. However, most of them overlook the time components, which are usually attached to the transactions stored in the databases. Thus existing algorithms cannot be applied to temporal databases directly as the temporal information is being ignored. Until recently, the problem of integrating temporal issues on association rules has been addressed [19, 65, 64].

The integration of *calendar* [15], which is a set of time intervals, into the discovery of association rules is proposed [65]. The concept of *calendric association rules* is defined where the rules found associate with the instances of the calendar. By segmenting the data over different time intervals, we may discover some interesting pattern which is previously ignored. For example, if beer and chips are sold together primarily between 6PM and 9PM on week days, when viewing the data over a week, we may not get enough support for such a pattern “beer \rightarrow chips (support:25%, confidence:75%)”. However, if we segment the data over two intervals, 7AM-6PM and 6PM-9PM, and consider only the data from weekdays, we may find that the support for the beer and chips rule in the segment 6PM-9PM jumps to 50%. Hence by analyzing data using a finer time granularity, we may find that some interesting rules exist only in certain time intervals but not occur in the whole period of time. Likewise, by determining any periodic intervals of some rules, the problem of mining *cyclic association rules* is introduced [55] which detects the periodical behavior of rules over time. For instance, if we find any association rules over monthly sales data, we may observe seasonal variation where certain rules are held at approximately the same in each year. However, this periodicity has limited power in describing real-life variations. Complicated patterns such as *the first working day of every month* cannot be described by simple periodic expression.

Similar idea with calendric association rules that segments the data into several intervals is suggested [19, 16, 18]. However, instead of gaining enough support to discover any interesting rules, the discovery of association rules with known valid periods and periodicities is concerned. It introduces the notion of temporal association

rules where absolute time is taken into consideration. We would find rules like “during summer, customers who buy bread and butter, also buy milk” if we consider the period between May and September instead of over a year. The temporal information extracted provides detailed information to reflect the dynamic changing data in reality rather than a static one. In particular, we may be interested in finding the longest interval that an association rule holds or behaves periodically [18].

On the other hand, by discovering any relationships between items which satisfy certain timing constraints such as *during*, *overlaps* in a temporal database, we might find some interesting associations between items recorded in the tuples [84]. We first group the tuples under certain timing constraint, we then examine the tuples in each of the group and find any association between groups of items. For example, if the duration of item A and item B overlap or intersect with each other, then A and B correlate.

The idea of adding temporal semantics to existing data mining techniques is raised in [64]. It extends the association rule mining technique to handle temporal semantics by examining the temporal relation between associated items. By first finding the associated attributes, we then look for any temporal relationships between them. The temporal nature of data is then captured. For instance, original association rules may tell us that `Investment_portfolio_X` is associated with `Insurance_policy_Y`. However, temporal associations may tell us that `Investment_portfolio_X` usually occurs after the start of `Insurance_policy_Y`. This may indicate the customers start with an insurance policy and becomes a gateway for other services such as `Investment_portfolio_X`.

2.5.3 Attribute-Oriented Induction

Another extension of work for accommodating temporal semantics into existing data mining techniques is attribute-oriented induction [63]. A temporal interval generalization framework (TIGF) to facilitate the generalization of time interval data is introduced. The integration of TIGF into existing algorithm for the induction of characterization rules is presented.

2.5.4 Time Series Analysis

Time series analysis focuses on symbolic patterns or numerical curve patterns in the sequences and is useful for many applications such as stock market data, financial data, telecommunication network data, etc. Besides analysing the change of shapes or before/after relationship, like mining of cyclic association rules, partial periodic

patterns of sequences is suggested [35]. By taking the partial period as an interval, methods for finding partial periodic patterns in time series are proposed. An example of a partial periodic pattern may state that “Jim reads the Vancouver Sun newspaper from 7:00 to 7:30 every weekday morning, but his activities at other times do not have such regularity”. Hence, instead of taking the whole sequence into consideration to look for any periodic patterns, partial periodic patterns is preferred. This idea of taking an interval of time of an event happens is similar to our problem which focused on interval-based events. However, unlike ours, periodicity instead of temporal relations among events is considered. Moreover, the length and timing of the interval for the partial periodic pattern are determined by the behavior of the repeating pattern while for us, the length and timing of the interval are based on the occurrences of the events themselves.

As most of the conventional data mining techniques do not accommodate interval data which is found to be useful in many applications, based on the previous work on temporal data mining especially mining sequential pattern, we extend the current work to accommodate interval data. In the following chapters, two interesting temporal patterns are introduced.

Chapter 3

Discovering Temporal Patterns for interval-based events

In this chapter, we describe the mining problem in more details. We aim to find temporal patterns, defined in terms of Allen's taxonomy of temporal relationships, in sequences of interval-based events stored in temporal database. We first formulate the concept of event sequence in Section 3.1 and introduce Allen's taxonomy of temporal relationships in Section 3.2. We introduce two temporal patterns, namely *AppSeq* and *LinkSeq* in Section 3.3. It is argued that both patterns are capable of expressing the complex relationships between interval-based events. An overview framework for mining such interesting patterns are given in Section 3.4. Finally, we summarize our discussion in Section 3.5.

3.1 Temporal Database

We adopt a discrete model of time, where each integer represents a point or instant in time upon the time-line. The granularity of time can be scaled between different segment sizes such as seconds and years, where the smallest possible granule size is defined as a chronon. The actual duration of a chronon is application specific.

Temporal database supports three types of time elements which are introduced in the previous chapter. They are namely *transaction time*, *valid time* and *user-defined time*. Such database captures the past and present data where the temporal attributes change values with time. Applications such as medical database, scientific databases in geophysics and astronomy, data are stored with the associated valid time. For instance, in health care database, patient record stores when the patient charge in and the time he/she discharge or any operation which takes a few hours to finish.

Such kind of data is stored in temporal database as historical records.

Each tuple associates a pair of ordered time points stating the period during which the information stored are valid. In our work, we focus on the valid time of the data stored as it indicates the evolution of data in time. All the events stored in temporal database happen in a linear time order where each event lasts for a period of time. We denote these events as *interval-based events*.

Here we describe our problem formally and introduce some terminologies used. Assume that in a temporal database, each database record contains a pair of ordered time points t_s and t_e where $t_s \leq t_e$ and both are positive integers. They are the **start time** and **end time** which together specify the valid time of the information stored. A record may have other attributes, for simplicity, we consider here a single temporal attribute and denote it as an **event**. We assume a set \mathbf{E} of event types.

Definition 1 [Event] *An event E has an associated time of occurrence and it is specified by a triple (A, t_s, t_e) , where $A \in \mathbf{E}$ is an event type and t_s and t_e are the start time and the end time, respectively. We also use $E.t_s$ and $E.t_e$ to indicate these times.* ■

Let us consider an example of a medical database. We are given a temporal database \mathcal{D} , each record in the database contains a person-id, name of disease and the start time and end time. Here, we suppose an event represents the contraction of a certain type of disease. In the general setting, we assume that each event is associated with one person. We assume the database is a set of **sequences**, each sequence consists of events for a particular person. There is at most one sequence for each person.

Definition 2 [Sequence] *A sequence of events is defined as a list of events where each event is associated with the same person: for person j , we have the following sequence s_j :*

$$s_j = \langle (A_1, t_{s_1}, t_{e_1}), (A_2, t_{s_2}, t_{e_2}) \dots (A_n, t_{s_n}, t_{e_n}) \rangle$$

The events are ordered by the end times where $t_{e_i} \leq t_{e_{i+1}}$ for all $i = 1, \dots, n - 1$. ■

This definition is similar to that of customer-sequence proposed in [8], where all the tuples associated with the same person can be grouped together to form a sequence. The events are ordered by the end time to ensure the former event ends on/before the following one as an order for interval-based events.


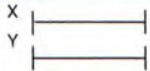
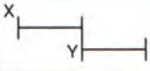
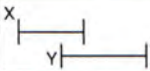
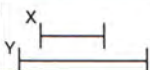

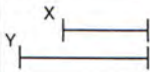
Relation	Symbol	Relation for Inverse	Pictorial Example	Endpoints Constraint
X before Y	b	Y after X		$X.t_e < Y.t_s$
X equal Y	e	Y equal X		$X.t_s = Y.t_s$ $X.t_e = Y.t_e$
X meets Y	m	Y met-by X		$X.t_e = Y.t_s$
X overlaps Y	o	Y overlapped-by X		$X.t_s < Y.t_s$ $X.t_e > Y.t_s$ $X.t_e < Y.t_e$
X during Y	d	Y contains X		$X.t_s > Y.t_s$ $X.t_e < Y.t_e$
X starts Y	s	Y started-by X		$X.t_s = Y.t_s$ $X.t_e < Y.t_e$
X finishes Y	f	Y finished-by X		$X.t_e > Y.t_s$ $X.t_e = Y.t_e$

Figure 3.1: The thirteen possible relationships between two intervals X and Y

3.2 Allen’s Taxonomy of Temporal Relationships

Relationship among time intervals can be described in different ways. **Allen’s taxonomy of temporal relationships** [9] is adopted to describe the basic relationships among events. It takes the notion of a temporal interval as primitive and obtains a set of temporal relations between intervals. As it provides a simple and natural syntax for specifying temporal relations between two intervals, the thirteen relationships are useful for describing the inter-relations between interval-based events. Figure 3.1 summarizes Allen’s thirteen temporal relationships. The relations between intervals can be expressed in terms of relations between their endpoints, we call this the *endpoint constraints*. For instance, consider the sequence $\langle E_1 = (A, 5, 10), E_2 = (B, 8, 12) \rangle$, we have “A overlaps B” since $E_1.t_s < E_2.t_s$, $E_1.t_e > E_2.t_s$ and $E_1.t_e < E_2.t_e$.

It is known that these thirteen relationships can be used to express any relationship held between two intervals and they provide a basis for the description of temporal patterns. Some of the relations are mirror image of the other, for example, “X overlaps Y” implies the same relation as “Y is overlapped-by X”. We only focus on seven primitive temporal relations with the order of items preserved. The seven relations are shown in the shaded area in Figure 3.1. Let us call this set of seven temporal

relations **Rel**. If $rel \in \mathbf{Rel}$ and events X and Y in a sequence have endpoints satisfying the constraints of $X \text{ rel } Y$, then we say that $X \text{ rel } Y$ is **true** in this sequence. For simplicity, we use symbols to represent the corresponding Rel in the figures of the following chapters.

As the endpoint constraints suggested, the order of events is important in the representation of temporal relations. We observe from the given seven primitive relations, for any two intervals X and Y , $(X.te < Y.te)$ or $(X.ts \geq Y.ts \text{ if } X.te = Y.te)$. Hence, in the following sections, we further restrict the order of the events as setting their end times in strictly ascending order while start times in descending order if equal end times.

We obtain a set of **binary predicates** if we consider any temporal relations between two events only. By combining the binary predicates, a sequence of events can be expressed as different **temporal patterns** \mathcal{P} . As we observe that, discovering all possible patterns can be computationally inhibitive and also the amount of results to the user can be overwhelming. We restrict our attention to the following two temporal patterns as they both provide simple and meaningful results showing the temporal behavior between events.

1. **AppSeq**: $((\dots(A_1 \text{ rel}_1 A_2) \text{ rel}_2 A_3) \dots \text{rel}_{k-1} A_k)$. We expands one event at a time and find the temporal relations between the preceding events to the following one. We may obtain pattern like “((A overlaps B) before C) overlaps D”. From the sequence, we get the idea of how the events are related to each other along the time-line. This is the first temporal pattern we are going to study and discuss in the next chapter.
2. **LinkSeq**: $((A_1 \text{ rel}_1 A_2) \& (A_2 \text{ rel}_2 A_3) \dots \& (A_{k-1} \text{ rel}_{k-1} A_k))$ where $\&$ is **AND** operation. We form the sequence by linking the common events in each binary predicate. Patterns like “(A overlaps B) $\&$ (B before C) $\&$ (C overlaps D)” are found. From the sequence, we obtain the individual temporal relations between two events clearly. It is the second temporal pattern we would investigate in Chapter 5.

We believe that both patterns are useful to describe the temporal behavior among events. They are simple and easy to understand. Besides knowing the ordering of events happen, we obtain a better understanding of how the events interact with each other. As both patterns are formed from binary predicates, they are closely related to each other or are different representations of a sequence. We first introduce the

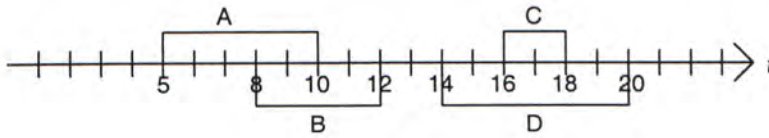


Figure 3.2: A sequence of interval-based events

framework for mining *AppSeq*. Then we continue to study the second pattern *LinkSeq* and describe it in details in Chapter 5.

3.3 Mining Temporal Pattern, AppSeq and LinkSeq

The first pattern is formed by combining the events one by one with the associated relations between the former group of events to the following one. The appending event has a greater end time than that of any preceding events. Consider a sequence shown in Figure 3.2, we combine the binary predicate “(A overlaps B)” with the following event “C” having the associated temporal relation as “before” by considering the endpoint constraints of the binary predicates and the following event. In result, we have the pattern “((A overlaps B) before C) overlaps D”. As the pattern formed by appending one event at a time, we call this **AppSeq**.

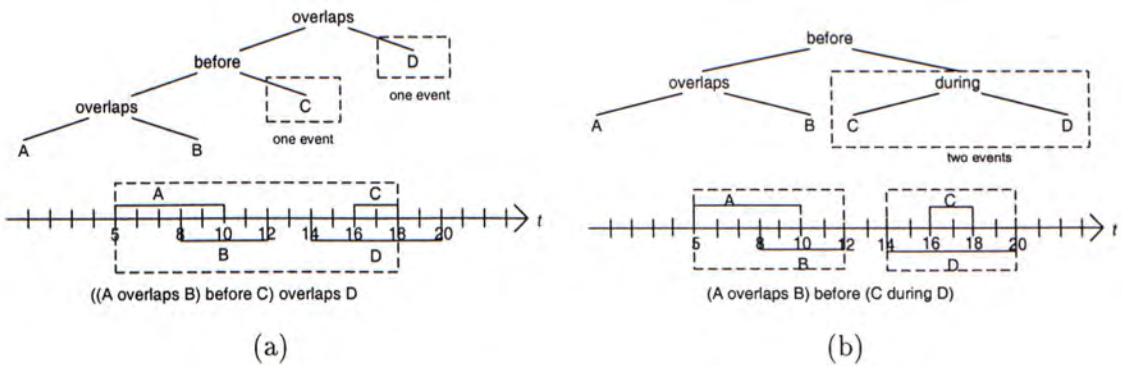


Figure 3.3: Different temporal pattern representation

3.3.1 A1 and A2 temporal pattern

Taking one step further, besides expanding patterns by only a single event, we may be interested in discovering temporal pattern where two patterns of sizes greater than one are combined via a temporal relation. We can see that in Figure 3.3, two different

patterns “((A overlaps B) before C) overlaps D” and “(A overlaps B) before (C during D)” hold in the sequence by considering different ways of combinations.

It is easy to see that the number of such possible descriptions is exponential in the sequence size. We restrict our interest to those patterns formed by appending one event at a time and we find temporal relations between a preceding group of events with the following one. That is, we consider temporal patterns of the form

$$((...(A_1 \text{ rel}_1 A_2) \text{ rel}_2 A_3)... \text{ rel}_{k-1} A_k)$$

We further call these the **A1 temporal pattern**. There are two main reasons for this restriction:

1. We believe that the temporal relations give some insight into causal relationships. As such, when a few events have happened, together they may become the cause of a following event. The temporal patterns in the above form gives us a modeling of this idea.
2. Discovering all possible patterns can be computationally inhibitive and also the amount of results to the user can be overwhelming. To verify the argument about the computation complexity, we have implemented the mechanism to discover temporal pattern obtained by appending composite pattern of size two at a time, it results in the patterns such as “(A overlaps B) before (C during D)”. We show by experiments that even a small extension in this way results in a much increased computational cost.

We name the other temporal pattern as **A2 temporal pattern**. Both A1 and A2 temporal patterns are derived from AppSeq, but they are slightly different in nature and we look for both of them. This leads to the differences between our mining methods which would be discussed in the next chapter.

3.3.2 Second Temporal Pattern, LinkSeq

One the other hand, our second temporal pattern known as *LinkSeq* is formed by linking two or more binary predicates with the common event as the binding point. Such a composition results in the form of “(A overlaps B) & (B before C) & (C overlaps D)” for the sequence shown in Figure 3.2. We extract the temporal knowledge by taking two events as a pair and investigate the relations between such binary predicates as a sequence.

LinkSeq looks similar to that of AppSeq in a sequence. We obtain both patterns by emphasizing two different temporal structure formed among events. In fact, AppSeq

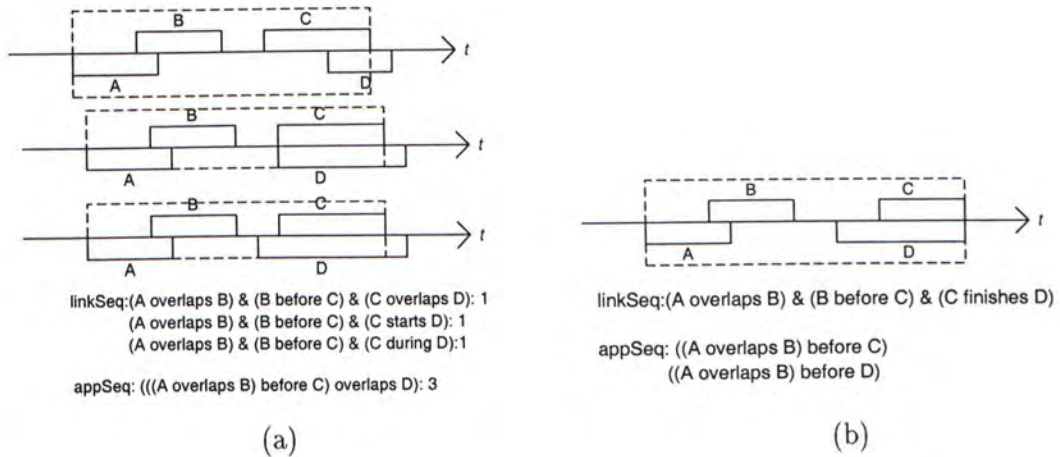


Figure 3.4: Differences between LinkSeq and AppSeq

is more general than that of LinkSeq. We illustrate this idea by giving an example. Consider the sequence shown in Figure 3.4(a). For AppSeq, we form the composite pattern “(A overlaps B) before C” which is going to append event D to form a new sequence. However, when we consider LinkSeq, the relation between C and D may not gain enough support to be discovered during the mining process. Hence the process of obtaining longer sequence of LinkSeq terminates. Unlike LinkSeq, AppSeq generalizes and groups some distinct relations as one relation which then gains enough support for the pattern. In result, more patterns can be found by AppSeq.

On the other hand, AppSeq expands one event at a time with the appending event having a greater end time than that of any preceding events. When we come across the temporal relation “finishes” or “equal” for the last two events of the sequence shown in Figure 3.4(b), we may only obtain “(A overlaps B) before C” and “(A overlaps B) before D” for AppSeq. However, for LinkSeq, we can obtain longer pattern as “(A overlaps B) & (B before C) & (C finishes D)” as we consider pairs of binary predicates. Hence LinkSeq includes the last event with equal end time as that of the preceding events. By considering the above two cases, we notice that each of the pattern reveals some temporal behavior of the events along the sequence and are closely related to each other. It is argued that both patterns are interesting and useful to describe the temporal behavior among interval-based events.

3.4 Overview of the Framework

As mentioned before, we pay special attention in mining two temporal pattern, AppSeq and LinkSeq. With the different formation of the patterns, we proposed

several methods using different strategies to tackle the problem. We here briefly describe our approach for mining both patterns.

3.4.1 Mining Temporal Pattern I, AppSeq

Regarding AppSeq, we form the pattern by appending one event at a time. Based on this idea, besides expanding one event, we examine a second pattern formed by appending two events (i.e. a binary predicate) each time. This results in an even number of events in the pattern which we denote it as A2 temporal pattern. It is actually an extension of the original pattern which we name it as A1 temporal pattern. Since both A1 and A2 temporal patterns are similar in formation, we first introduce the method, **AppOne**, by considering the simplest form of AppSeq as shown in Figure 3.3(a). We then further extend the method for finding the other kind of pattern as shown in Figure 3.3(b) as the second method, **AppTwo**.

A basic strategic similar to the Apriori-like approach [7] is used and using a layout, which we call *item-list*, in the mining process for both AppOne and AppTwo. For comparison, a performance study for both methods through experiments on synthetic data is conducted.

In addition, as both patterns are formed from binary predicates, we observe that both methods suffer from the high cost of generation of binary predicates during the early stage of the mining process. To deal with this problem, we further investigate an improved mechanism that we can form the binary predicates directly from data without undergoing any candidate generation phase. We would discuss the methods in details in Chapter 4.

3.4.2 Mining Temporal Pattern II, LinkSeq

With different formation of our second temporal pattern, LinkSeq, we extract the temporal knowledge using different data structures which results in several methods to perform the mining task.

First, intuitively, we may find LinkSeq by deriving from AppSeq. As all the frequent events which form the AppSeq must form the corresponding binary predicates with each other, we can form LinkSeq by linking the binary predicates with the order of events preserved which is already inherited in AppSeq. However, as mentioned in the previous section, AppSeq may miss the last event if its end time is the same as that of the preceding one. We may not find the complete set of LinkSeq as it suggests. Despite of this, the method is useful when we consider both LinkSeq and

AppSeq together as result. We denote this method as **LinkApp**.

As the formation of LinkSeq from combining the binary predicates suggests, we can modify the previous AppOne method to accommodate the needs for the second pattern, using the Apriori-like approach. We call this **LinkTwo** method.

Besides using an Apriori-like approach which divides the mining process into candidate generation phase and support counting phase, inspired by the recent work on mining frequent pattern [37], we propose to use a tree-like structure, namely *seq-tree* to store the sequence data. By exploring the compact information stored in the seq-tree, we develop an efficient mining method for the pattern LinkSeq as **LinkTree**.

We would discuss these alternatives for mining LinkSeq in Chapter 5. Experimental results showing the performance of these methods are presented also.

3.5 Summary

In this chapter, we introduce the two temporal patterns, called AppSeq and LinkSeq. Unlike most previous approaches of mining sequential patterns which considers only point-based events, we consider interval-based events, which we believe are important in many applications. We employ Allen's taxonomy of temporal relationships as a mechanism to express temporal patterns between interval-based events. We discover that the number of temporal relationships can be prohibitively large and also many of such patterns may be complicated and of little value to the users. We therefore consider two types of temporal patterns which is simple and meaningful. We describe the mining methods for the two patterns in details in Chapter 4 and 5.

Chapter 4

Mining Temporal Pattern I, AppSeq

In this chapter, we discuss our first temporal pattern, AppSeq, which further subdivides into A1 and A2 introduced previously. We start by giving the notion of temporal pattern in the following section. Methods for mining the A1 and A2 pattern are described in Section 4.2 and 4.3 respectively and a modified approach proposed in Section 4.4. Experimental results are presented in Section 4.5. Finally, Section 4.6 summarize our discussion on mining AppSeq.

4.1 Problem Statement

As mentioned in Chapter 3, we assume each event E has an associated time of occurrence as $E.t_s$ and $E.t_e$. We obtain binary temporal predicates if we consider two events only. To express complex relation among more events in a sequence, we form A1 temporal pattern by appending one event at a time via a temporal relation and have following definition:

Definition 3 [Temporal Pattern, A1] *A temporal pattern is defined recursively as follows:*

- *If E is a single event type in \mathbf{E} , then E is a temporal pattern, it is also called an **atomic pattern**.*
- *If X and Y are two temporal patterns, and $rel \in \mathbf{Rel}$, then $(X \text{ rel } Y)$ is also a temporal pattern. This is a **composite pattern**.*

*The **size** of a temporal pattern is the number of atomic patterns in the pattern. ■*

Based on this definition, we have “A” and “B” as atomic patterns for the sequence shown in Figure 3.2. The sizes of all atomic patterns are the same as 1. Composite patterns include “A overlaps B” and “((A overlaps B) before C) overlaps D” with size of pattern being 2 and 4 respectively. We mainly focus on finding any composite pattern which consists of a group of atomic patterns.

Definition 4 [Mapping] *An atomic temporal pattern X has a **mapping** in a sequence S if we can find an event E of type X in the sequence. We denote this mapping by $\mathcal{M}(X, S) = \{E\}$. We associate a time duration to the mapping as follows: $\mathcal{M}(X, S).t_s = E.t_s$ $\mathcal{M}(X, S).t_e = E.t_e$ We say that X **holds** in S .*

*A composite pattern $(X \text{ rel } Y)$ in which Y is an atomic pattern and $\text{rel} \in \mathbf{Rel}$ has a **mapping** $\mathcal{M}((X \text{ rel } Y), S)$ in a sequence S if X has a mapping $\mathcal{M}(X, S)$ in S and we can find an event $E \notin \mathcal{M}(X, S)$ of type Y in S to be mapped as $\mathcal{M}(Y, S)$ such that if we consider some imaginary event Z with start time of $\mathcal{M}(X, S).t_s$ and end time of $\mathcal{M}(X, S).t_e$, then $Z \text{ rel } E$ is true.*

*In this case, $\mathcal{M}(X \text{ rel } Y, S) = \mathcal{M}(X, S) \cup \{E\}$. We say that the relation $(X \text{ rel } Y)$ **holds** in S . The mapping $\mathcal{M}((X \text{ rel } Y), S)$ has an associated time interval given by*

$$\mathcal{M}((X \text{ rel } Y), S).t_s = \min\{\mathcal{M}(X, S).t_s, \mathcal{M}(Y, S).t_s\}$$

$$\mathcal{M}((X \text{ rel } Y), S).t_e = \mathcal{M}(Y, S).t_e$$

In the above mapping of a composite pattern in a sequence, **union** of two time intervals takes place. We form a minimum time interval that includes the events in the composite pattern. The resultant interval is determined by the minimum of the start times and the maximum of the end times of X and Y respectively. For example, for the sequence shown in Figure 3.3(a), we obtain a composite pattern “(A overlaps B) before C” with the resultant interval being [5,18].

Moreover, user can specify the maximum length of time interval that is of interest, known as the **window-size**, win_size . The intuition is that if some events do not happen close enough to each other, we would not be interested to find any correlation between them.

Definition 5 [window-size] *If w is a given window-size, a temporal pattern P **holds within** w in a sequence S if there is a mapping $\mathcal{M}(P, S)$ such that $\mathcal{M}(P, S).t_e - \mathcal{M}(P, S).t_s \leq w$.*

Definition 6 [*k*-item] Let $A_i, i = 1, \dots, k$ be a bag of k event types in \mathbf{E} , $rel_i \in \mathbf{Rel}$, $i = 1, \dots, k-1$, a *k*-item has the form

$$\{\{A_1, A_2, \dots, A_k\}, \{rel_1, rel_2, \dots, rel_{k-1}\}, \mathcal{P}\}$$

where \mathcal{P} is a temporal pattern in terms of the events types A_1, A_2, \dots, A_k and the relations rel_1, \dots, rel_{k-1} , and $k \geq 1$. ■

Given a window-size w , let M be a subset of the set of events in a sequence S , M **supports** an *k*-item $\{\{A_1, A_2, \dots, A_k\}, \{rel_1, rel_2, \dots, rel_{k-1}\}, \mathcal{P}\}$ if M is a mapping for the temporal pattern \mathcal{P} and $M.t_e - M.t_s \leq w$. We also say that each event in M supports \mathcal{P} . For example, if the window-size is 100, the given sequence in Figure 3.2 supports both the temporal pattern “((A overlaps B) before C) overlaps D” and “(A overlaps B) before (C during D)”.

The **support** of a temporal pattern \mathcal{P} in a set of sequences \mathcal{D} is defined as the total number of different mappings in all sequences in \mathcal{D} for the pattern over the total number of sequences in \mathcal{D} . i.e.,

$$support(\mathcal{P}, \mathcal{D}) = \frac{|\{M \subseteq S | S \in \mathcal{D}, M \text{ supports } \mathcal{P}\}|}{|\mathcal{D}|}$$

The **support** of a *k*-item is the support of the temporal pattern in the *k*-item.

A **large *k*-item** is a *k*-item having support greater than a threshold, namely *min_sup* provided by users, that is, $support(\mathcal{P}, \mathcal{D}) \geq min_sup$. Our aim is to find the large *k*-items for all k .

4.2 Mining A1 Temporal Patterns

Here we propose a method, AppOne, for mining frequent A1 temporal pattern as shown in Figure 3.3(a). Let us use an example to illustrate how the method works. The example is a patient database of the form shown in Table 4.1. Each tuple contains a person-id, the disease contracted by the patient and the duration of the disease. The database can be used to find if some diseases are likely to cause some other diseases and their temporal relations. It is assumed that the minimum support is 33% and the window-size is set to be 30 time units.

We use a layout of event sequence that is different from the one used in finding sequential pattern [8]. Instead of transforming the original database into a list of sequences, the **seq-list**, where each sequence corresponds to one person, we use an **item-list** to represent the temporal data. Each event is associated with a list of

person-id	disease	start	end
1	A	5	10
1	B	8	12
1	C	20	24
1	E	17	28
1	F	14	28
1	B	28	32
2	A	12	20
2	B	16	22
2	C	24	28
2	E	28	31
2	F	27	31
2	G	33	35
3	A	8	12
3	I	8	12
3	D	8	18
4	D	7	12
4	A	17	22
4	J	22	24
4	G	15	26
5	D	10	15
5	A	17	22
5	G	15	26
6	A	14	18
6	B	16	22
6	C	32	36
6	E	28	38
6	F	26	38

Original database

person-id	seq-list
1	(A,5,10),(B,8,12),(C,20,24), (E,17,28),(F,14,28),(B,28,32)
2	(A,12,20),(B,16,22),(C,24,28), (E,28,31),(F,27,31),(G,33,35)
3	(A,8,12),(I,8,12),(D,8,18)
4	(D,7,12),(A,17,22),(J,22,24), (G,15,26)
5	(D,10,15),(A,17,22),(G,15,26)
6	(A,14,18),(B,16,22),(C,32,36), (E,28,38),(F,26,38)

seq-list

item	pid-list
A	(1,5,10),(2,12,20),(3,8,12), (4,17,22),(5,17,22),(6,14,18)
B	(1,8,12),(1,28,32),(2,16,22), (6,16,22)
C	(1,20,24),(2,24,28),(6,32,36)
D	(3,8,18),(4,7,12),(5,10,15),
E	(1,17,28),(2,28,31),(6,28,38)
F	(1,14,28),(2,27,31),(6,28,38),
G	(2,33,35),(4,15,26),(5,15,26)
I	(3,8,12)
J	(4,22,24)

item-list

Table 4.1: Transform the database as *seq-list* and *item-list*

person-id, start time and end time (pid, t_s, t_e) . Table 4.1 illustrates the differences between the two approaches using the above example of patient records. Note that each item is an atomic or composite pattern and the t_s and t_e in the item-list indicates an time interval for this pattern. Similar idea of transforming the database into the form which we called item-list is used in [85].

A basic strategic similar to the Apriori-gen approach [7] is used. With seq-list, we need to scan the database once in each iteration. The item-list approach avoids this problem since it enables us to count support by direct composition of the lists. The size of these lists and number of candidates shrink as the sequence length increases, which facilitates fast computation. This motivates us to choose item-list format to store the large k -items for efficient support counting.

Initially, we compute the large 1-items in a single database scan by storing the large atomic patterns into an item-list. Let us refer to the set of large k -items as L_k . We then generate candidates by combining a $(k-1)$ -item in L_{k-1} , with a single event in L_1 . A k -**candidate** is of the form $\{A, B\}$ where A is a $(k-1)$ -item and B is an 1-item. For a given database, let us refer to the set of all k -candidates generated by our method as C_k .

A major task in the generation of large k -items is to determine if any k -candidate is contained in a sequence. For each candidate, we examine the L_{k-1} and L_1 item-lists and determine the temporal relations between the composite pattern and atomic pattern that have sufficient supports. We then generate new large k -items and obtain the L_k item-list by merging the composite and atomic patterns with the temporal relation. The composition of two item-lists to form L_k is depicted in Figure 4.1.

We only need to scan the database once to create the L_1 item-list in the first pass. For further iterations, we simply join the item-lists to obtain the large k -items. Hence the computation time mainly depends on the size of the item-lists for L_k and C_k which would shrink for later iterations. The main algorithm is shown in Figure 4.2. The algorithm terminates when we cannot find any large k -items after the end of the current pass. The discovery process is mainly divided into the following phases:

Sort phase: We first group the records with person-id as the major key and end time as the minor key for sorting in ascending order while start time in descending order as shown in Table 4.1.

Initial phase: During the first pass, we determine the large atomic patterns called 1-items, which would be used repeatedly for forming composite patterns in later iterations. We store the large 1-items as item-list as shown in Table 4.2.

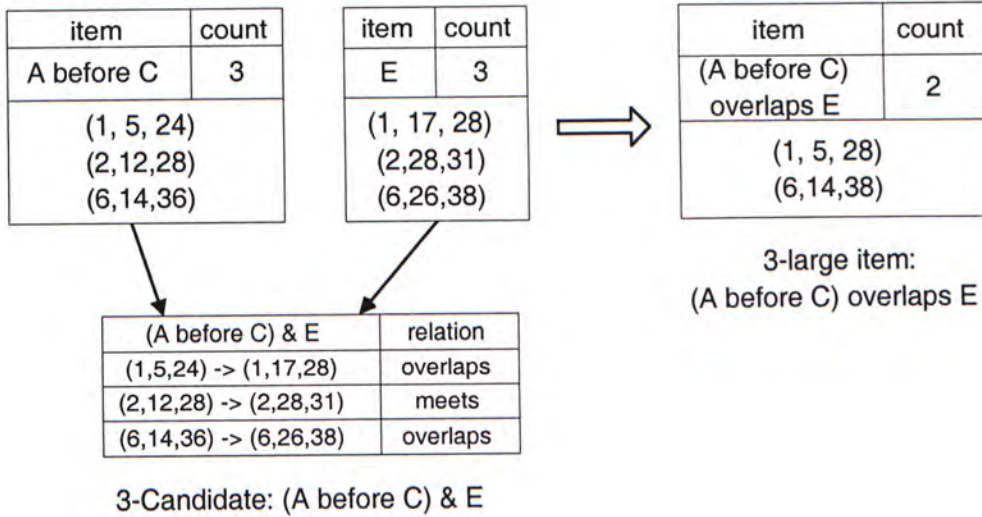


Figure 4.1: Composition of two item-list, L_2 and L_1

item	count	item	count	item	count	item	count	item	count
A	6	B	4	C	3	D	3	E	3
(1,5,10)		(1,8,12)		(1,20,24)		(3,8,18)		(1,17,28)	
(2,12,20)		(1,28,32)		(2,24,28)		(4,7,12)		(2,28,31)	
(3,8,12)		(2,16,22)		(6,32,36)		(5,10,15)		(6,28,8)	
(4,17,22)		(6,16,22)							
(5,17,22)									
(6,14,18)									

Table 4.2: Partial Large 1-item list

Candidate generation: In each subsequent pass, we add one large atomic pattern to a composite pattern in the L_{k-1} set to form a new potentially large item, such k -items are called the candidates. Details are described in the next subsection.

Large k -items generation: After generating the candidates, we scan the item-list for counting the support for the candidate. i.e. the number of sequences that support the temporal patterns within the window. At the end of the k -th pass, the algorithm determines the large k -items. We would further elaborate the mechanism of generating large items in Section 4.2.2.

4.2.1 Candidate Generation

We use an array to store the candidates. The logical form of the candidates is shown in an example in Table 4.3. The candidate generation to obtain C_k from L_{k-1} is done

Main Algorithm Input : A set of atomic patterns. Output: All large k -items in $\cup_k L_k$

Algorithm 4.1 Main Algorithm

```

1   $L_1 = \{1\text{-large items}\}$  //tuples containing items with minimum support
2  for ( $k=2$ ;  $L_{k-1} \neq \emptyset$ ;  $k++$ ) do
3    If  $k=2$  then generate  $C_2$  from  $L_1$  (Refer to section 4.2.1)
4    else  $C_k = \text{candidate\_gen}(L_{k-1}, L_2, L_1)$ 
5     $L_k = \text{large\_items}(C_k)$  (Refer to section 4.2.2)
6  end
    
```

Figure 4.2: The main algorithm

item 1	item 2
A overlaps B	C
A overlaps B	E
A overlaps B	F
A before C	E
A before C	F
B before C	E
B before C	F
⋮	⋮

Table 4.3: The 3-candidates

by adding one large 1-item each time. We generate the candidates by combining the events in atomic patterns with those in composite patterns. For the first set of candidates, we consider all the combinations from L_1 to form C_2 . Since our binary temporal pattern of the form $A \text{ rel } B$ implies that the end time of B is at least as late as that of A , we make sure that this is true when we generate a candidate of the form $\{A, B\}$. In the following passes, large k -items are formed for $k \geq 2$. Such k -items are composite patterns. We generate C_k from L_{k-1} . In this case, knowledge regarding the temporal relations between the composite patterns and the atomic patterns is applied. We would prune any irrelevant candidates in this phase.

The algorithm for the candidate generation is given in Figure 4.3, in which we describe the pruning step. For instance, for the 2-item with a pattern of “C during E” shown in Table 4.4, we aim to find any temporal relation between the 2-item and event A. In the previous pass, we have found that no pattern of the form “C *rel* A” or “E *rel* A” is large, we therefore exclude the possibility of having the candidate of “(C during E)” with “A” in the generation of large 3-item. Table 4.3 shows the

Input : L_{k-1}, L_2, L_1 (where $k > 2$)

Output: C_k (candidate set)

Algorithm 4.2 candidate_gen(L_{k-1}, L_2, L_1)

```

1 for each pair of composite patterns  $b_i \in L_{k-1}$  and atomic patterns  $a_j \in L_1$  do
2   Let  $b_i$  be the  $k - 1$ -item with  $((..(a_{i1}rel_{r1}a_{i2})...rel_{rk-2}a_{ik-1})$  where  $rel_{ri} \in Rel$ 
3   if  $(a_{i1}rel_{ij}a_j) \in L_2$  or  $(a_{ik-1}rel_{ij}a_j) \in L_2$  for any  $rel_{ij}$  then
4     Generate the candidate element  $\{b_i, a_j\}$ 
5   end
6 end
    
```

Figure 4.3: The candidate generation algorithm

item	count	item	count	item	count	item	count
A overlaps B	3	A before C	3	B before E	3	C during E	2
(1,5,12)		(1,5,24)		(1,8,28)		(1,17,28)	
(2,12,22)		(2,12,28)		(2,16,31)		(6,28,38)	
(6,14,22)		(6,14,36)		(6,16,38)			

Table 4.4: Partial large 2-item list

generation of 3-candidates.

Lemma 1 Algorithm 4.2 generates all the potentially large k -items.

Proof In Algorithm 4.2, we try to form a k -candidate from one $(k - 1)$ -item and an 1-item. Let X be the $(k - 1)$ -item and Y be the 1-item. Let event type a_{i1} be the first event type in X and a_{ik-1} be the last event type in X . In the pruning, we make sure that either a_{i1} or a_{ik-1} can form a pattern with Y by one of the seven relations and the pattern has support $\geq min_sup$. Suppose that $(X rel Y)$ has support $\geq min_sup$. There are seven relations in **Rel**. Let us consider each of them

1. X before Y : if X before Y holds in a sequence then each of the events mapped to X will be before those mapped to Y .
2. X equal Y : if X equal Y holds in a sequence then this means the starting time of the event mapped to a_{i1} in X is equal to the starting time of the mapping to Y and hence the relation of *starts* is true for them.
3. X meets Y : if X meets Y holds in a sequence then this implies that the event mapped to a_{ik-1} in X meets the mapping of Y .

4. X overlaps Y : if X overlaps Y holds in a sequence then the event that is mapped to a_{ik-1} in X may overlaps/starts/during with the mapping of Y .
5. X during Y : if X during Y holds in a sequence then all the events mapped to X have the *during* relation with the mapping of Y .
6. X starts Y : if X starts Y holds in a sequence then the event mapped to a_{i1} in X starts the mapping of Y .
7. X finishes Y : the event with the latest end time in X finishes Y .

Therefore in all cases, X, Y will be generated as a k -candidate. ■

4.2.2 Large k -Items Generation

This phase is further divided into two subphases. They are the support counting phase and the generation of large items.

Support Counting: First, we need to find the supports for the candidates generated. We determine the number of sequences that support each temporal relation of the composite pattern in each candidate. We compare the endpoint values of elements in L_{k-1} and L_1 and determine if any temporal relation holds between the composite and atomic items. Large k -items are formed if their support is greater than the threshold given.

To facilitate efficient counting, we use a hash tree to store L_1 and also the relevant part of the 1-item-list and a hash tree to store L_{k-1} and part of the $(k-1)$ -item-list. We use the value of the event as a key for hashing in the hash tree for L_1 . For the hash tree for composite patterns, we use the values of all the events included and the temporal relations together to form a key by simple concatenation. The leaf nodes of the hash tree corresponds to some large k -item I and it also records the mappings for the pattern in I . The mappings are stored in the form of item-list, with the person-ids, start times and end times. For each candidate, we use the composite pattern and event as search keys and find from the hash trees the corresponding L_{k-1} and L_1 items.

During the search in the hash tree, hashing is done in the internal nodes until we reach the leaf nodes where we perform exact matching for the composite or atomic pattern. We consider a pattern P for a large $(k-1)$ -item and a pattern P' for a large 1-item. We can identify any temporal relation that holds between a mapping for P and a mapping in P' , since the start times and end times are recorded in

Input: C_k

Output: L_k

Algorithm 4.3 large_items(C_k)

```

1  for each candidate  $c \in C_k$  do
2    for each relation  $rel_i \in Rel$  do
3       $c.count\_rel_i =$  support count of  $c$  with respect to  $rel_i$  from  $L_{k-1}, L_1$ 
4      satisfying the win_size threshold.
5      if  $c.count\_rel_i \geq min\_sup$  then
6         $L_k = L_k \cup (b_i \ rel_i \ a_j)$  where
7           $b_i \in L_{k-1}$  and  $a_j \in L_1$ ,  $b_i$  and  $a_j$  are the elements of  $c$ ,
8           $(b_i \ rel_i \ a_j).t_s = \min(b_i.t_s, a_j.t_s)$  and
9           $(b_i \ rel_i \ a_j).t_e = a_j.t_e$ 
10     end
11   end
12 end

```

Figure 4.4: Support counting for the candidates

the corresponding hash trees. If some composite pattern is found, the count for the candidate with respect to the specific temporal pattern is increased by 1. The counts are kept as auxiliary information to the table for the candidate items. There are seven counts for each candidate, one for each of the temporal relationship.

Forming Large k -Items: The second subphase is to form large k -items. Table 4.4 shows a partial set of the large 2-items. After identifying any temporal relation between the items in C_k , we generate L_k from L_{k-1} and L_1 with the corresponding temporal relation. Each new item in L_k is a composite pattern and is used for the next pass. The resultant interval is obtained from the union of two intervals of $(k-1)$ -item and 1-item. For instance, as shown in Table 4.4, the start time and end time of the mapping of composite pattern “A overlaps B” are $\{[5,12], [12,22], [14,22]\}$. The algorithm for forming the large items is summarized in Figure 4.4.

Lemma 2 *Algorithm 4.1 generates all large k -items.*

Proof: If a k -item I is large, then its support is at least min_sup , and in any sequence that supports the I , the set of events E mapped to the temporal pattern in I would appear within an interval that is smaller than the user specified window size threshold. Therefore, for a subset D of E , the support is at least as great as that of E , and the events would also appear within an interval that is smaller than the window size

threshold. All such subsets D has been considered in the candidate generation, and hence we are guaranteed to discover all large k -items. ■

Lemma 3 *Every large k -item generated by Algorithm 4.1 represents a frequent temporal pattern.*

Proof: Based on the formation of large k -item, we generate a resultant interval of the composite pattern from the union of two intervals of $k - 1$ -item and 1-item. Hence, the resultant interval obtained should be the minimum interval that includes both $k - 1$ -item and 1-item, i.e. the associated time of occurrence of k -item formed. On the other hand, for the support counting phase in k -th iteration, we determine any temporal relations between $k - 1$ -item and 1-item by examining their endpoint values. As the end time of the 1-item is equal to or greater than that of k -item, by using the endpoint constraints shown in Figure 3.1, we obtain the corresponding temporal relations having enough support to form large items. In result, we obtain the temporal relation between preceding events, $k - 1$ -item, with the appended event, 1-item, as AppSeq. Hence every k -item formed generates a corresponding frequent temporal pattern. ■

With the lemmas above, we show that the method, AppOne, correctly finds the complete set of frequent A1 temporal patterns. As we can see from the mining process, there are only two scans of the original database. In the formation of large items, composition of two item-lists takes place for each candidate. Thus the main cost of the above method is the composition of item-lists, $|L_{k-1}|$ and $|L_1|$, which is determined by the size of candidate set, $|C_k|$ and $|L_1|$. As mentioned before, the size of the item-lists are comparatively smaller than that of the original database and the size of candidate set would shrink in later iterations. Hence, at the early stage of the mining process, more time and memory are spent on the generation of large number of L_k .

4.3 Mining A2 Temporal Patterns

In the previous section, in mining the A1 temporal pattern, we generate large k -items by adding one atomic pattern in L_1 at a time. Here we consider a slightly more complex form of temporal patterns which we call A2 temporal pattern. The A2 temporal pattern is defined recursively as the following: (1) a temporal pattern of size 2 is an A2 temporal pattern. E.g. "A overlaps B". (2) if X is an A2 temporal pattern,

item	count	item	count	item	count
(A overlaps B) before (C during E)	2	(A before C) overlaps (B before E)	2	(A overlaps B) before (E finishes F)	3
(1,5,28) (6,14,38)		(1,5,28) (6,14,38)		(1,5,28) (2,12,31) (6,14,38)	

Table 4.5: Partial large 4-item list formed by AppTwo

and Y is a temporal pattern of size 2, then $(X \text{ rel } Y)$ where $\text{rel} \in \mathbf{Rel}$ is also an A_2 temporal pattern. Example of such a composition is shown in Figure 3.3(b). The patterns we generate are in even number of events, i.e. $2k$ -items. We therefore focus on temporal relations among events by adding one 2-item each time. By modifying the previous method, AppOne, to accommodate the adding of 2-item in this case, we derive method method, called AppTwo, for mining A_2 temporal pattern. In fact, AppTwo works similarly as AppOne, except for the candidate generation phase and the formation of large $2k$ -items.

4.3.1 Candidate Generation:

In the formation of C_2 , the process is the same as before. Next we start to generate C_{2k} , where $k \geq 2$, we examine L_{2k-2} and L_2 and use compositions of the elements in the two item-list of L_{2k-2} and L_2 . When we prune any irrelevant candidates in this phase, we need to consider two newly added atomic patterns this time, say a_{j1} and a_{j2} , where $a_{j1} \text{ rel }_j a_{j2} \in L_2$. The two added items can be combined with a composite pattern, say b_i , where $b_i \in L_{2k-2}$ if both of the following conditions hold:

1. there is a relation between the leftmost atomic pattern of b_i and at least one of a_{j1} and a_{j2} .
2. there is a relation between the rightmost atomic pattern of b_i and a_{j1} or a_{j2} .

For example, consider the 2-item “A overlaps B” shown in Table 4.4. Since we can find “A before C” and “B before E” as large 2-items, we then include the combination of “A overlaps B” with “C during E” as one of the 4-candidates. The candidate generation algorithm for the second method is shown in Figure 4.5.

Lemma 4 *Algorithm 4.4 generates all the potentially large $2k$ -items.*

Input : L_{2k-2}, L_2 (where $k \geq 2$)

Output: C_k (candidate set)

Algorithm 4.4 candidate_gen_second(L_{2k-2}, L_2)

```

1  for each pair of composite items  $b_i \in L_{2k-2}$  and composite item  $b_j \in L_2$  do
2      Let  $b_i$  be the  $2k - 2$ -item with  $(..(a_{i2}rel_{r_2}a_{i4})...rel_{r_{2k-4}}a_{i2k-2})$  where  $rel_{r_i} \in Rel$ 
3      and  $b_j$  be the 2-item with  $(a_{j1}rel_{j1}a_{j2})$ 
4      if  $[(a_{i2}rel_{ij}a_{j1}) \in L_2$  or  $(a_{i2}rel_{ij}a_{j2}) \in L_2$  and
5       $(a_{i2k-2}rel_{ij}a_{j1}) \in L_2$  or  $(a_{i2k-2}rel_{ij}a_{j2}) \in L_2]$  for any  $rel_{ij}$  then
6          Generate the candidate element  $\{b_i, b_j\}$ 
7      end
8  end

```

Figure 4.5: The candidate generation algorithm of AppTwo

Proof In Algorithm 4.4, $2k$ -candidates are generated from one $2k - 2$ -item and one 2-item. Let X be the $2k - 2$ -item and Y be the 2-item. a_{i2} has the earliest start time in X and a_{i2k-2} has the latest end time in X . a_{j1} has the earliest start time in Y and a_{j2} has the latest end time in Y . In the pruning, we make sure that both a_{i2} and a_{i2k-2} have at least one of the seven relations with either a_{j1} or a_{j2} in Y . Suppose that X is related to Y . There are seven relations in Rel. We again consider each of them

1. X before Y : if X is before Y then all the events in X will be before Y .
2. X equal Y : this means the starting time of the event a_{i2} in X is equal to the starting time of the a_{j1} in Y and the ending time of event a_{i2k-2} in X is equal to the ending time of the a_{j2} in Y and hence a_{i2} starts a_{j1} and a_{i2k-2} finishes a_{j2} .
3. X meets Y : this implies that a_{i2} in X before a_{j1} in Y and a_{i2k-2} in X meets a_{j1} in Y .
4. X overlaps Y : in this case, the starting time of event a_{i2} in X before starting time of a_{j1} in Y and the ending time of event a_{i2k-2} in X before the ending time of a_{j2} in Y and hence a_{i2} (before/meets/overlaps) a_{j1} and a_{i2k-2} (before/meets/overlaps/during/starts) a_{j2} .
5. X during Y : the starting time of event a_{i2} in X before the ending time of event a_{j2} in Y and the ending time of event a_{i2k-2} in X before the ending time of

event a_{j2} in Y and hence a_{i2} and a_{i2k-2} (before/meets/overlaps/during/starts) a_{j2} .

6. X starts Y : the event with the earliest start time in X starts a_{j1} in Y and the ending time of event a_{i2k-2} in X before ending time of event a_{j2} in Y and hence, a_{i2} starts a_{j1} and a_{i2k-2} (before/meets/overlaps/during/starts) a_{j2} .
7. X finishes Y : the starting time a_{i2} in X before ending time of event a_{j2} in Y and the event with the latest time in X finishes a_{j2} in Y and hence, a_{i2} (before/meets/overlaps) a_{j2} and a_{i2k-2} finishes a_{j2} .

Therefore in all cases, X, Y will be generated as a $2k$ -candidate. ■

4.3.2 Generating Large $2k$ -Items:

We also divide this into two phases namely support counting and the generation of large items. In general, the second method works in the same manner as the first method in that we generate incrementally larger $2k$ -items by combining the two composite patterns of a $2k$ -candidate. The difference is that we shall use the item-list of L_{2k-2} and L_2 .

We observe that some patterns formed by the combination of L_2 are quite complex, not very natural and not easily understandable. Table 4.5 shows the 4-items for the above example. As we can see, one of the 4-items, “(A before C) overlaps (B before E)” is ambiguous to represent the relation between events. To deal with this, we further restricts our A2 temporal pattern with those temporal relations between L_{2k-2} and L_2 to *before* and *meet* only. In results, we can obtain meaningful and interesting temporal patterns. In our example, “(A overlaps B) before (C during E)” can be obtained.

4.4 Modified AppOne and AppTwo

As we use an Apriori-like approach for generating A1 and A2 temporal pattern in our methods, we suffer from the high cost of handling a huge number of candidate sets in the first few iterations, especially, C_2 , as no pruning strategy is applied. To deal with this problem, we try to use another way to generate L_2 without C_2 generation. A tree-like structure storing all the information for all binary predicates is proposed. By traversing the tree once, we can obtain the list of L_2 directly from data. The method runs as follows.

1. A scan of database derives a list of L_1 which are potential candidates for forming large sequences.
2. We scan the database again. For each sequence, we examine every pair of 1-items and form a set of binary predicates. Using the above database shown in Table 4.1, let us take the first sequence as an example, we obtain the binary predicates as shown in Figure 4.6. After scanning all the sequences, we obtain the complete set of binary predicates which are potential candidates for forming L_2 .
3. We then construct a tree of depth being two using the binary predicates collected. For each distinct binary predicate, a branch is created by taking the parent node being the precedent event and the child node being the second event. We store the temporal relation, the corresponding count, pid, start and end time as a list associated with the child node. If there exist a branch representing the binary predicate, then increment the corresponding count by 1 and insert the pid, start and end time to the associated list. The tree formed by scanning the first sequence of the given database is shown in Figure 4.6.
4. By traversing the whole tree once for each child from the root, we obtain L_2 by giving out the branches having counter values greater than threshold.

As we can see, all the binary predicates are generated from L_1 for each sequence, they are in fact potential candidates for forming L_2 as both events of the binary predicate are frequent events. By further examining the corresponding support for the temporal relations between the pair of events which has been already stored in the tree, we obtain the list of L_2 in result.

For further iterations, we form L_k using the previous two methods for mining A1 and A2 temporal patterns. The only difference is the formation of L_2 which no candidate is formed in this phase. We can see that the size of tree formed depends solely on the number of binary predicates. We observe that the maximum size of the tree is $|L_1| \times |L_1| \times |Rel|$ which may greater than $|C_2|$. However, we suppose the number of binary predicates formed is far less than that of $|C_2|$, by generating all possible cases, the formation of L_2 without C_2 is expected to be more efficient. This assumption holds when repetition of the binary predicates is high such that less distinct binary predicates are formed. We obtain **modified AppOne** and **modified AppTwo** by applying the technique to both AppOne and AppTwo respectively. As a comparison, we study the performance of all these methods in the next section.

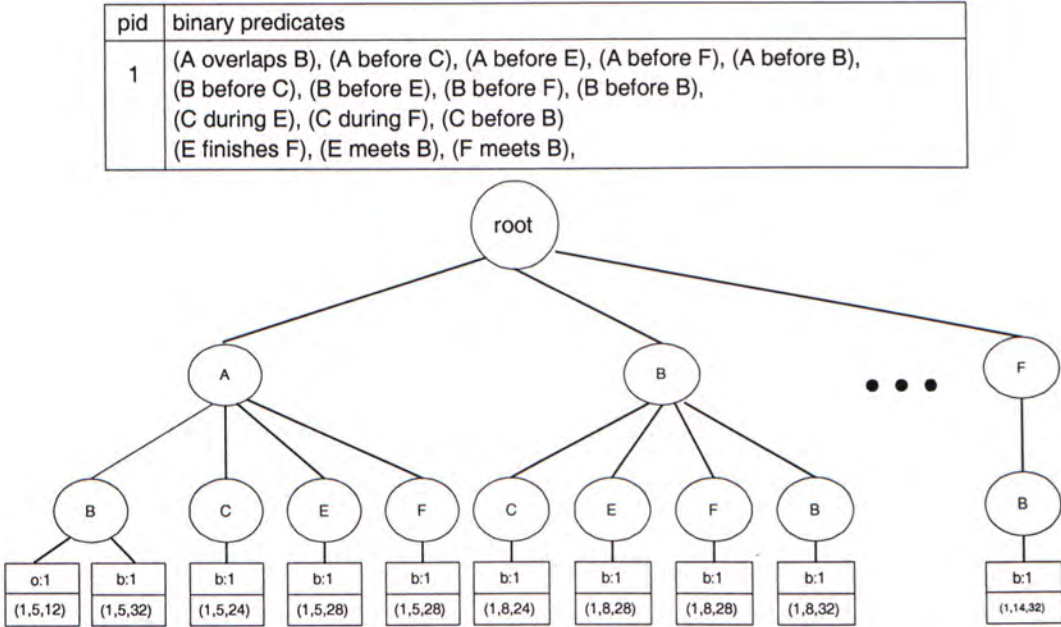


Figure 4.6: Forming L_2 without C_2

$ D $	Number of sequences
$ P $	Average number of events per large item
N_S	Number of maximal potentially large item
N	Number of event types

Table 4.6: Parameters

4.5 Performance Study

To evaluate the performance of the proposed methods over a large range of data, we conducted several experiments on an UltraSparc 5/270 workstation with 520MB of main memory. All methods in are written in C. We consider synthetic data in an application domain of a medical database same as that of the given example. For each person we record a sequence of clinical records stating the different diseases contracted.

4.5.1 Experimental Setup

The synthetic data generation program took the parameters shown in Table 4.6. The data generation model was based on the one used for mining sequential pattern [8] with some modification to model the patient database. We first formed a table T of large items in which the number of items was set to N_S . We generated each large

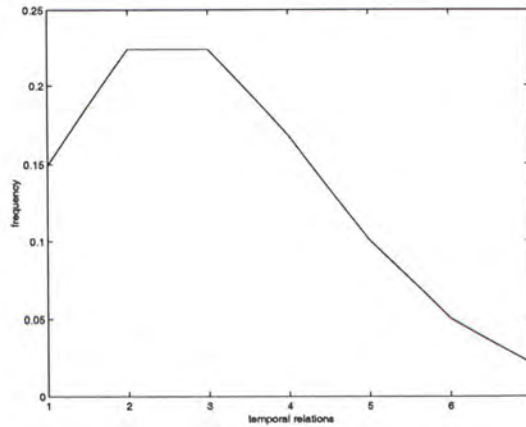


Figure 4.7: Distribution of temporal relations between events

item by first picking a number of events from a Poisson distribution with mean μ equal to $|P|$ and we chose the event types randomly. We then picked a temporal relation between events and formed a pattern. Temporal relations are chosen from the set **Rel**. We generated patterns that contain the seven temporal relations according to a distribution shown in Figure 4.7. We used the values of $\{1,2,3,4,5,6,7\}$ to represent *starts*, *overlaps*, *before*, *during*, *finishes*, *meets*, *equal* respectively. The distribution of Figure 4.7 was determined arbitrarily by our intuitive expectation of the likeliness of the relations. Each person was then assigned a potentially large item which was chosen from the table T of items. The time interval of each event followed an exponential distribution with mean μ equal to 5. For each item, the time where the first event took place was chosen randomly within the time interval $[0,500]$ of time units. The following events of the item then followed the temporal relation held between events. For the temporal relation *before*, the time where events were separated followed an exponential distribution with mean μ equal to 5. For the relation *overlaps*, the time interval where two events overlaps was restricted to an exponential distribution with mean μ equal to 3. For the *during* relation, the time that the latter event delays under an exponential distribution with mean μ equal to 3. We generated the dataset by setting $N_S=2000$, $N=1000$, $|D|=10K$ and $|P|=5$ with 1MB of data. We studied the effect of different values of *min_sup*, *win_size*, number of sequences and events per sequence, etc., for the two methods.

4.5.2 Experimental Results

First, we studied the effect of minimum support on the processing time. We used 7 values of minimum support (*min_sup*) as shown in Figure 4.8, and 100 time units for

window size (*win_size*) for the test. Figure 4.8 shows the decrease of the execution time when the minimum support increases for both patterns. As the support threshold increases, less large items are generated and hence the size of the candidate set in each iteration decreases dramatically. The execution time would thus decrease for less time is required for support counting and hash tree searching of large items. Comparing the two methods, AppOne and AppTwo, AppTwo needs much more time than that of AppOne. This is due to large amount of computation time in pruning the candidates as the addition of two atomic patterns are considered instead of one. We observe that both modified AppOne and modified AppTwo did help a little in the mining process, especially for large support threshold. They generate L_2 without forming C_2 at the early stage of the process. However, in the following iterations, the high cost of candidate generation still dominate in the mining process, especially for small support threshold as more iterations are involved.

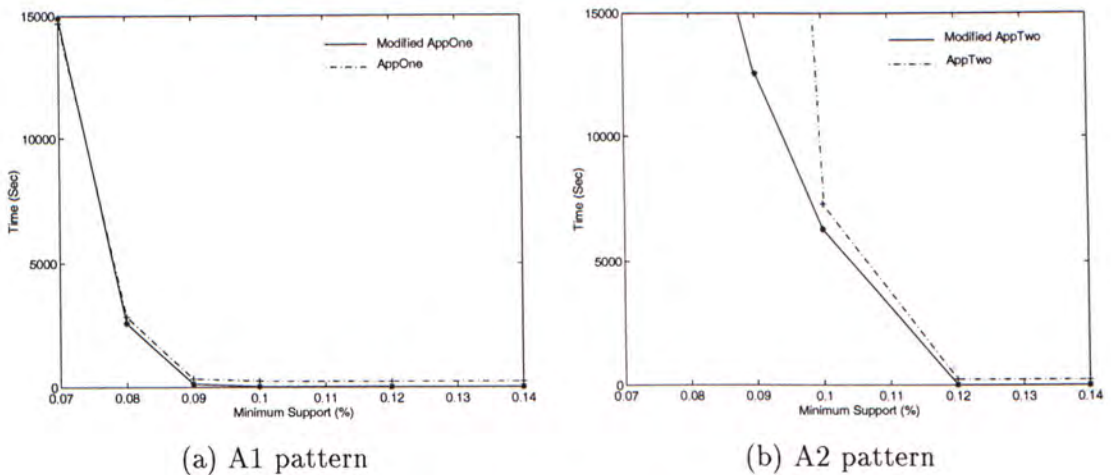


Figure 4.8: Variation on minimum support

Figure 4.9 shows the execution time at each pass for finding two patterns. We set the $min_sup = 0.0008$ for A1 temporal pattern and $min_sup = 0.001$ for A2 temporal pattern. For AppOne, from the fifth pass onwards, much less time is needed. As more large items are formed at the first few passes, the support counting of L_k where $k \leq 5$ dominates the execution time. However, for further iterations, as the size of the item-list shrinks, the support counting process is much faster. However, AppTwo does not behave like AppOne. It ran with fewer passes but took more time in the candidate pruning and support counting phase of L_4 as there is a very large C_4 being generated. Both the modified AppOne and AppTwo use little time in the second pass to form L_2 . Since we use the iterative Apriori-like approach for the following passes, the execution time used in the following passes is more or less the same as

that of AppOne and AppTwo. The number of large items generated in each pass is shown in Figure 4.10. We set $min_sup = 0.0008$ for AppOne and $min_sup = 0.001$ for AppTwo. The number of large items generated by AppOne is greater than that of AppTwo when compare with the same length of patterns. i.e. $|L_4|$ of AppOne is compared with $|L_4|$ of AppTwo.

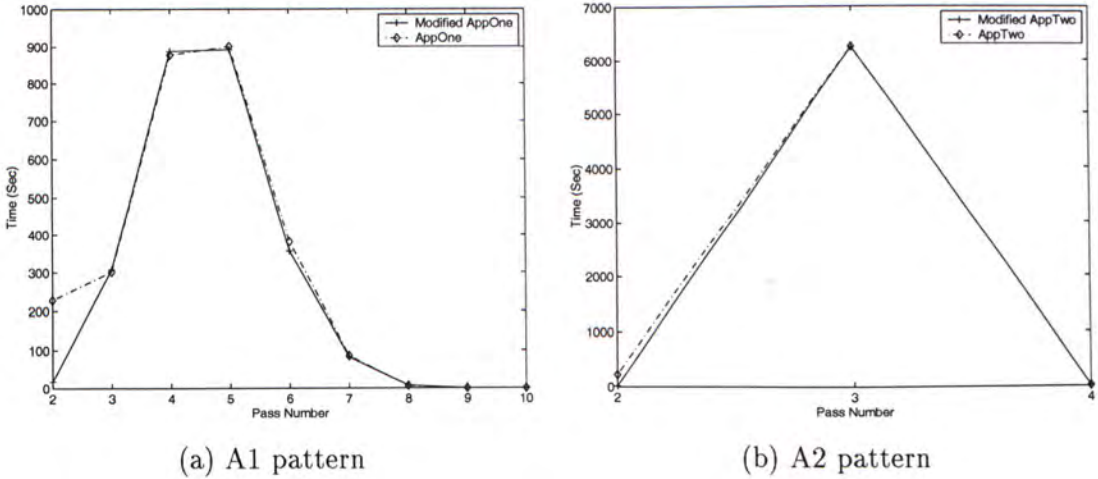


Figure 4.9: Execution time for each pass

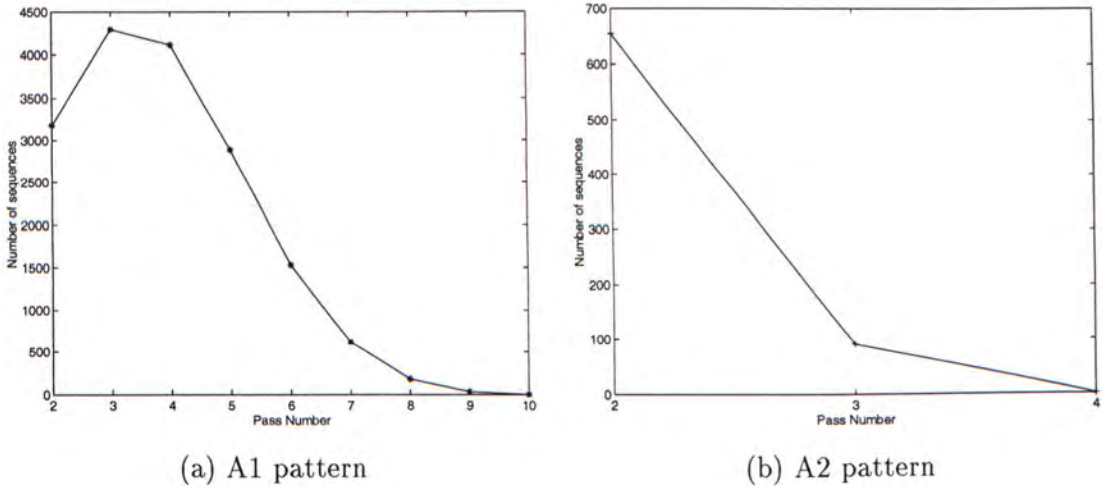


Figure 4.10: Number of large items generated in each pass

We then studied the effect of window size on the processing time. We chose the values of min_sup being 0.0008 for AppOne and modified AppOne, 0.001 for AppTwo and modified AppTwo. In Figure 4.11, we can see that when the window size increases, the execution time increases for both patterns. This is because more sequences are included and the time for support counting phase increases. Also the number of iterations increases and which also requires a longer execution time. Both

min_sup	no. of resulting seq.	max. seq. length
0.0007	30872	10
0.0008	16836	10
0.0009	5946	9
0.0010	2600	9
0.0012	153	6
0.0014	19	4

Number of A1 patterns

min_sup	no. of resulting seq.	max. seq. length
0.0009	1574	4
0.0010	746	4
0.0012	83	3
0.0014	16	2

Number of A2 patterns

Table 4.7: Number of AppSeq with different *min_sup*

the modified AppOne and modified AppTwo work better than that of the original methods since the time for generating L_2 is reduced.

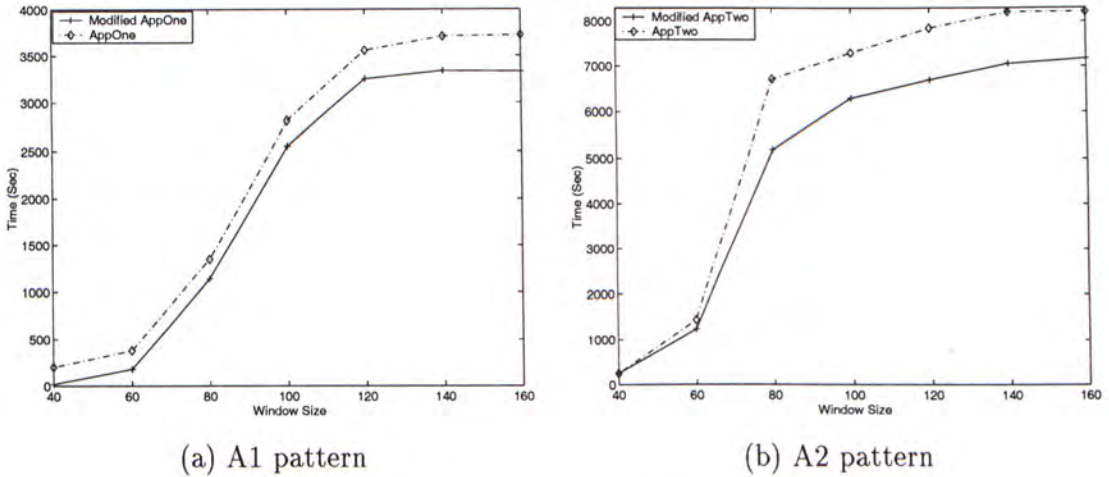


Figure 4.11: Variation on window size

Table 4.7 shows the number of AppSeq obtained with various values of *min_sup* and *win_size* is set to be 100 time units. For both patterns, the number of patterns decreases rapidly as the *min_sup* increases. However, the number of A2 pattern is far less than that of A1 pattern. This may be due to the complex structure of A2 pattern so that less number of patterns can be formed.

Scale-Up Experiment

Our next target is to consider the scale-up effects. We examined how the performance varies with the number of sequences. Figure 4.12 shows how the methods scale up as the number of sequences is increased ten-fold, ranging from 10K to 100K and with *min_sup* = 0.0025 for both patterns. The execution time for AppOne and

AppTwo increase with increasing number of sequences. However, regarding modified AppOne and modified AppTwo, since large number of L_1 and L_2 are generated, the approach of generating L_2 from data improves the performance greatly.

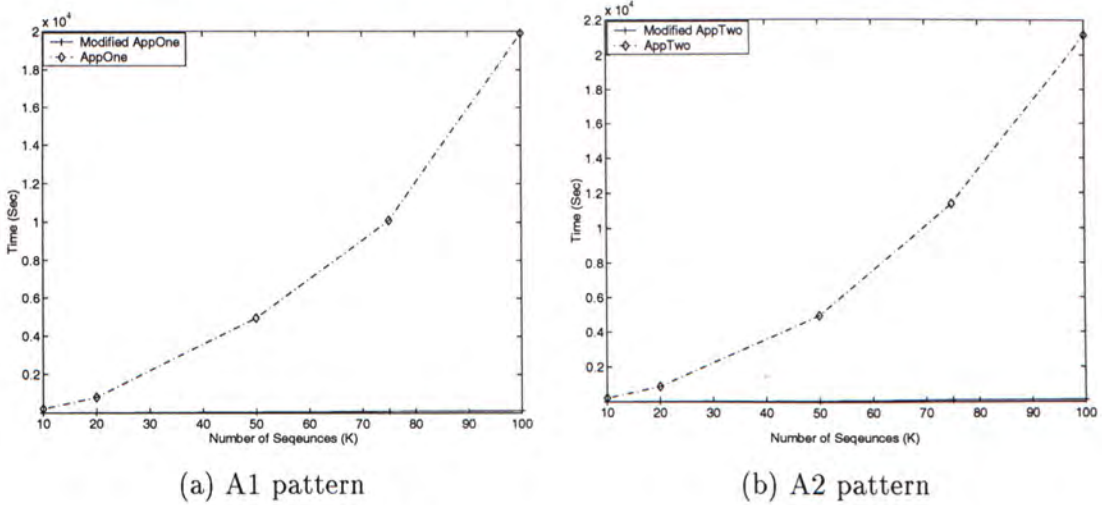


Figure 4.12: Scale-up: Number of sequences

We finally studied the scale-up as we increase the average number of events per sequence. The number of sequences used is 10K and kept constant. We vary the average number of events per sequence from 2.5 to 25. Figure 4.13 shows the scalability results of the methods. We set $min_sup = 0.0025$ for the four methods. From the figure, the execution time increases rapidly with the increasing number of events per sequence. As the average number of events per item increases, items with longer patterns are formed. Then more large items are formed and it results in more candidates being generated in each iteration. This increases the computation time for the hash tree searching for support counting dramatically. Moreover, as items with longer pattern are formed which results in a greater number of iterations, the execution time is thus lengthened. From the figure, both modified AppOne and modified AppTwo outperform AppOne and AppTwo respectively.

4.5.3 Medical Data

For our real-life data experiment, we use a data set which contains clinical records of Scoliosis patients, i.e. patients who suffer from spinal deformation. A scoliosis patient may have one or several curves in his/her spine. Among them, the curves with severe deformations are identified as major curves. In this experiment, we aim to find any frequent temporal patterns, A1 temporal pattern, in the data set and found the corresponding common features they cover. The database stores about 900 patients

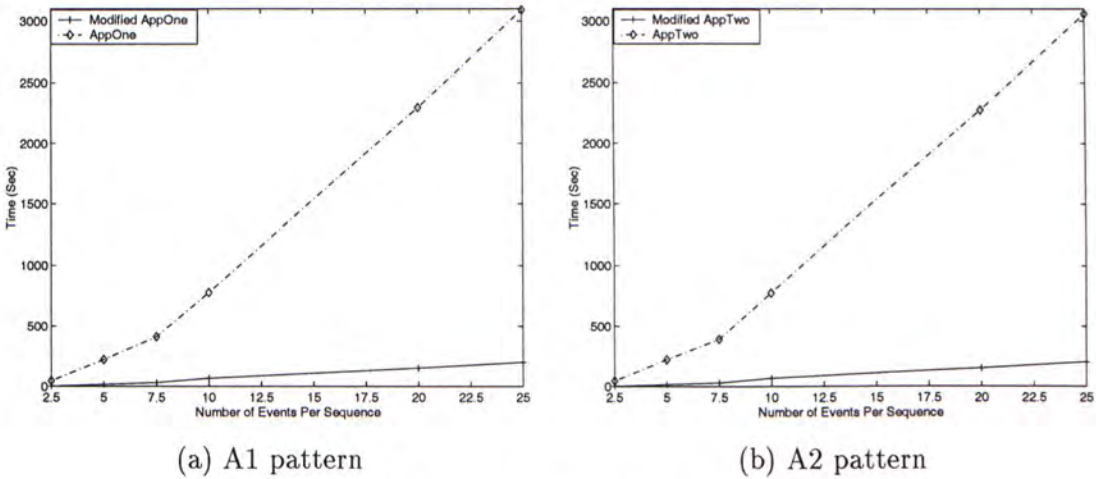


Figure 4.13: Scale-up: Number of events per sequence

where each record stores person-id, date of birth, family history, class of scoliosis contracted and treatment. It also stores measurements on the patients, such as the number of curves, the curve locations, degree of curvature and direction of each curve, etc.

Most of them are numerical data which are temporal in nature such as age, height, weight and degree of curvature. Some of them are categorical and non-temporal data. We pay special attention to the spine curvature values as they reflect the level of spine deformation. With consecutive records of each visit, a sequence of curvature values are captured for each patient. By examining the changes in values in the view of a sequence, we may discover any temporal knowledge stored in the database.

We start by eliminating any missing data in the database as each angle measured of the spine is crucial for diagnosis. Then we extract some of the useful and essential attributes in finding frequent sequence. We further partition the curvature degrees into different interval values to find any common pattern like the way for mining quantitative data [75]. For each patient, we obtain at most four curvature values for different parts of the spine for each visit. We map the curvature values into an event in our model by simple concatenation. For example, a patient's spine has four curve degrees which are 10.0, 20.0, 22.0, 0.00 with different directions and location. We first match the corresponding numerical values into different intervals say interval 1, 2, 2, 0 if we take 10.0 degree as an interval range. We then represent the data as an event in our model as "1220". Our intuition is that, each spine is characterized by combining the four degrees as a whole, together with the curve direction and location. We therefore regard the four curvature values as the feature of a spine. Then we represent the data as a single event. Obviously, we can form tremendous

different types of events depending on the segment for each interval values of the curvature degrees. We keep the person-id to distinguish the sequences. And we transform the number of visit as the ordering time by taking the start time and end time being the number of visit instead of using the actual visit date for simplicity. Moreover, for consecutive visits with no change in the curvature values, we merge the records by setting the end time being the last visit. Hence we obtain a sequence of both points and interval-based events for we take each event representing different spine deformation .

We use AppOne to find any interesting temporal pattern in this database. As we would like to view the whole treatment process for each patient, we set the *win_size* as the largest number of visit stored in the database. Table 4.8 presents the statistics of finding large temporal pattern in the scoliosis database with various *min_sup*. Since we look for any sequence with two or more visits involved, we found from the database that only 463 patients have two or more visits. Thus when we talk about *min_sup*, we suppose to focus on the 463 potential sequences stored. The number of large items decreases rapidly as the *min_sup* increases. As the values for curvature degrees are diverse and the distribution are sparse, not many common sequence can be found and most of them are short in length, terminated in L_2 . However, from the practical point of view, we do not expect to find too complex temporal relation as they are difficult to be interpreted. The results obtained depends on the way in partitioning the curvature values into intervals that a wide range of values may grouped into the same category. As we look for the changes for the spine deformation, we would like to classify any similar spines while distinguishing any special ones. Thus expert knowledge of any indication of curvature values may help in the partition and better results can be obtained. The common sequences found can be used with other attributes such as the type of scoliosis, treatment, family history, etc., stored in the database for further investigation. One of the results found is as follows: “3100 before 3200 \rightarrow isIS , conf(0.7), sup(0.022)” which is read as “70% of patient whose spine deforms from category 3100 to 3200 (changes in the second part of the spine) are determined as having Idiopathic Scoliosis. The results may lead us to have some interesting findings from the data.

4.6 Summary

In this chapter, we propose several methods for discovering interesting temporal patterns, AppSeq. We consider a special type of temporal pattern which is simple and

<i>min_sup</i>	no. of sequences	iterations	total time(s)
0.009	42	4	0.27
0.011	28	3	0.25
0.015	15	3	0.19
0.019	7	3	0.16
0.024	5	3	0.15
0.030	0	2	0.09

Table 4.8: Mining AppSeq in a scoliosis database

meaningful. We show that an iterative method with the flavor of the Apriori-gen function can be used for the mining. With the use of the item-list format, we can perform fast and simple composition of temporal patterns for both candidate generation and support counting. A set of experiments has been conducted to demonstrate the overall performance of the methods.

For comparison we consider the mining of a slightly more complex type of temporal pattern, A_2 which is a modification of the original temporal pattern A_1 . From experiments, we find that the computation time required for the first pattern, A_1 , is much more acceptable. On the other hand, the approach of generating L_2 without C_2 do help in some of the cases since no compositions of item-lists is needed. This approach especially favors for the case when large support threshold is held.

Moreover, we use a real set data containing clinical records of Scoliosis patients. We are interested in finding any A_1 temporal pattern within the data. AppOne is used for the mining purpose and we obtain some experimental results which may need further examination with expert knowledge for interpretation.

For the mining method, we here propose an Apriori-like approach forming large number of C_k during the intermediate steps. We would investigate other data structure as well as algorithms which help to reduce the high cost in candidate generation of the mining process.

Chapter 5

Mining Temporal Pattern II, LinkSeq

We introduce our second temporal pattern, known as *LinkSeq*, in this chapter. We give the notion of temporal representation of LinkSeq in Section 5.1. Methods for mining the temporal patterns are given in Section 5.3 and 5.4 using different data structures to facilitate efficient support counting process. Experimental results showing the performance of the methods are presented in Section 5.5. Finally, we have a summary in Section 5.7.

5.1 Problem Statement

As discussed previously in Chapter 3, we form another pattern by linking the binary predicates with the common event as the binding point. Such composition results in the form of $((A_1 \text{ rel}_1 A_2) \& (A_2 \text{ rel}_2 A_3) \cdots \& (A_{k-1} \text{ rel}_{k-1} A_k))$. We extract the temporal knowledge by taking two events as a pair and investigate the relations between the events in the binary predicates as a sequence. According to the formation of the pattern, LinkSeq is defined as follows:

Definition 7 [Temporal Pattern, LinkSeq] *A temporal pattern is defined recursively as follows:*

- *A temporal pattern of size 2 of the form $(X \text{ rel } Y)$, i.e. binary predicate, is a LinkSeq*
- *If X is a LinkSeq, and Y is a binary predicate with a common event type A_{k-1} such that $X = ((A_1 \text{ rel}_1 A_2) \& (A_2 \text{ rel}_2 A_3) \cdots \& (A_{k-2} \text{ rel}_{k-2} \underline{A_{k-1}}))$ and $Y = (\underline{A_{k-1}} \text{ rel}_{k-1} A_k)$, then $X \& Y$ is also a LinkSeq*

The **size** of a temporal pattern is the number of distinct events in the pattern. ■

With this definition of temporal pattern, we have “(A overlaps B) & (B before C) & (C during D)” as one of the Linkseq formed for the sequence of Figure 3.2 with size equals to 4. We focus on the temporal relations between each pair of events by adding one binary predicate each time. The idea of window-size and support defined in the previous section is also applicable here with little modification as follows.

Definition 8 [window-size] *If w is a given window-size, a temporal pattern P holds within w in a sequence S if $A_k.t_e - A_1.t_s \leq \text{win_size}$ where A_1 is the first event in a temporal pattern and A_k is the last event.* ■

Definition 9 [k -item] *Let $A_i, i = 1, \dots, k$ be a bag of k event types in \mathbf{E} , $rel_i \in \mathbf{Rel}$, $i = 1, \dots, k-1$, a k -item has the form*

$$((A_1 \text{ rel}_1 A_2) \& (A_2 \text{ rel}_2 A_3) \& \dots (A_{k-1} \text{ rel}_{k-1} A_k))$$

An event sequence support k -item if the temporal pattern holds in the event sequence. The **support** of a temporal pattern \mathcal{P} in a set of sequences \mathcal{D} is defined as the total number of occurrence of the patterns over the total number of sequences \mathcal{D} . i.e.,

$$\text{support}(\mathcal{P}, \mathcal{D}) = \frac{|\{\mathcal{P} \text{ holds within } S | S \in \mathcal{D}\}|}{|\mathcal{D}|}$$

The **support** of a k -item is the support of the temporal pattern in the k -item.

A **large k -item** is a k -item having support greater than a threshold, namely min_sup provided by users, that is, $\text{support}(\mathcal{P}, \mathcal{D}) \geq \text{min_sup}$.

5.2 First Method for Mining LinkSeq, LinkApp

LinkSeq looks similar to that of AppSeq in a sequence. We obtain both patterns by emphasizing two different temporal structure formed among events. As we observe that we may find LinkSeq by deriving from AppSeq, we have the following mechanism.

1. We first obtain L_k of AppSeq with all the frequent events involved.
2. For each L_k , we derive the corresponding LinkSeq by examining every pairs of events within the pattern. We do so by looking up the list of L_2 which is found previously and follow the order of events to link them up as a sequence.

The intuition is that, if the frequent events form the AppSeq, they must form large items as L_2 during the process. As in the formation of AppSeq, the order of events are preserved and thus can form the LinkSeq also. One point to note is that AppSeq excludes the case when the last event embedded in the pattern has same end time as the preceding events. In other words, shorter sequence is formed. Despite of this, we can derive LinkSeq easily if AppSeq is already given. As this approach forms LinkSeq from AppSeq, we denote the method as **LinkApp**. In Section 5.5, we show our results which we find LinkSeq from AppSeq using LinkApp.

5.3 Second Method for Mining LinkSeq, LinkTwo

Another way to tackle the mining problem is to modify AppOne which is introduced in Section 4.2 in the previous chapter, to accommodate the needs for the second pattern. As the formation of LinkSeq suggests, no mapping of start time and end time of composite pattern is needed. We obtain LinkSeq by linking up the binary predicates with the common event as the binding point and keeping the orders of the binary predicates along the sequence. Hence we come up with the following method, namely **LinkTwo**, as it links up two events at a time. LinkTwo works similarly as AppOne except for the candidate generation phase and the formation of large k -items.

Candidate Generation: In the formation of C_2 , the process is the same as before. However, as we found in the previous chapter, we can form L_2 without C_2 using the approach developed for modified AppOne, we would choose this approach for generating L_2 for LinkSeq also.

Next we start to generate C_k , where $k \geq 3$, we examine L_{k-1} and L_2 and use compositions of the elements in the two item-list of L_{k-1} and L_2 . We prune any irrelevant candidates by examining any common event between L_{k-1} and L_2 , say $a_{j1} \text{ rel}_j a_{j2} \in L_2$ where a_{j1} is the common event. The two added items can be combined with a LinkSeq, say b_i , where $b_i \in L_{k-1}$ if it has the common event a_{j1} as the last event in the pattern. For example, consider the 2-item shown in Table 4.4, since we can find “A before C” and “C during E” as large 2-items, we then include the combination of “(A before C) & “(C during E)” as one of the 3-candidates.

Generating Large k -Items: In general, LinkTwo works in the same manner as AppOne in that we generate incrementally larger k -items by combining two patterns, L_{k-1} and L_2 , of a k -candidate. However, the formation of large k -items would be much simple than that of AppOne. For support counting phase, we shall use the

item-list of L_{k-1} and L_2 and we examine the *pid* attribute only to ensure all events are associated with the same person as a sequence. In the second sub-phase to form large k -items, no union of start time and end time of the two patterns is needed as we simply use $\&$ to link up the binary predicates. No further investigation of the temporal relations is performed.

5.4 Alternative Method for Mining LinkSeq, LinkTree

For the above methods, we use an Apriori-like approach which divides the mining process into candidate generation phase and support counting phase. We use item-list to facilitate the checking of *pid* for each candidate. As we can see, for LinkSeq, no examination of start time and end time is required. We then argue that the relative order of events provides sufficient information for mining LinkSeq, rather than the start time and end time as that for AppSeq. Nevertheless, when we transform the database into item-list, we may not keep the order of events properly. Hence instead of using item-list in our mining process, inspired by the idea of *frequent pattern tree*, or **FP-tree** for short, suggested in [37] recently for mining frequent patterns [7], we propose to use a tree-like structure to store the sequence data, namely **seq-tree**. In Chapter 2, we have introduced the structure of a FP-tree and the key idea of the mining algorithm. As FP-tree suggested, it avoids generating a huge set of candidates by keeping useful information and storing it into a compact data structure. No further scanning of original database is needed for later mining process. Experimental results show that the use of FP-tree for mining frequent pattern outperforms previous Apriori-like method. This motivates us to investigate a tree-like structure that captures all essential information for the mining process. As for LinkSeq, we focus on the relative order of binary predicates which basically depends on the start time and end time of each event, we propose to use seq-tree to store the relative order of binary predicates and reduce the high cost of support counting for large candidate set. Hence we have the following alternative method for mining the temporal pattern and denote as **LinkTree**.

5.4.1 Sequence Tree: Design

We design the structure of a seq-tree with the following observations. First, the relative order of events is a crucial information in determining the temporal patterns. Hence, we use the end time and start time to determine the order of nodes to be

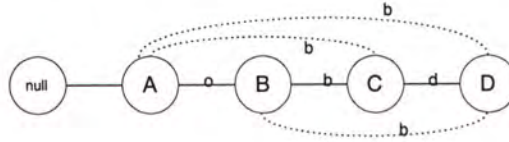


Figure 5.1: A seq-tree constructed for a sequence

placed along the paths of the tree. We consider the former event as the parent node and the latter event as the child node for each binary predicate.

Secondly, the number of occurrences of each binary predicate is recorded as *count* which counts the number of sequences that include the binary predicate. With the count kept in the seq-tree, by traversing the tree once, we obtain a list of frequent binary predicates in a sequence by examining the counter values. No further scanning of database is needed.

Thirdly, for any frequent events which appear in the same sequence, the frequent events are placed along the same path from the root node of the tree. This ensures the correctness of the frequent patterns found as all the events along the same path comes from any sequence that contains the same sequence of binary predicates. No generation of candidates and scanning of database for support counting are required and hence must less costly for mining the temporal pattern.

In addition, as our problem deals with complex temporal structure other than before/after, more complex structure of the seq-tree is required to capture the essential information to form LinkSeq. Let us consider the sequence shown in Figure 3.2. At the first glance, we obtain the longest temporal pattern as “(A overlaps B) & (B before C) & (C during D)”. We store such pattern by **main branch** (or **main-bh**) in our seq-tree. However, other patterns such as “(A before C) & (C during D)”, “(A before D)”, etc., are also interesting to be found. We then store such patterns by **subsidiary branch** or (**sub-bh**). The resultant seq-tree is shown in Figure 5.1. In fact, both main-bh and sub-bh are same in structure. They both are used to store binary predicates. We divide the set of binary predicates into these two groups for the construction of seq-tree described later. We here further introduce several terminologies for the following discussion.

Definition 10 [main branch] *Main branches (or main-bh in short) are defined as a sequence of binary predicates which forms the longest temporal pattern.* ■

Definition 11 [subsidiary branch] *Subsidiary branches (or sub-bh) are defined as a set of binary predicates which shows the temporal relations between any two events of*

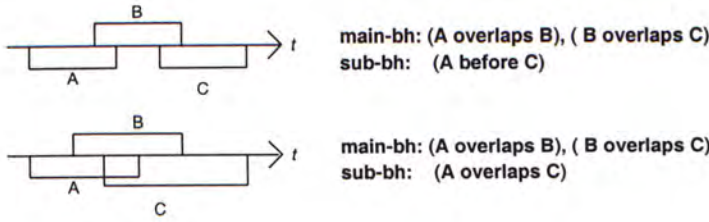


Figure 5.2: Identical Pattern if same main-bh and sub-bh

a sequence other than those stored in the main branch. ■

A branch is actually an edge between two nodes in the seq-tree. While main branches form the skeleton of the tree, subsidiary branches store other temporal relations between any two events which are useful for mining the complex structure of the temporal pattern.

Main branches basically reveal the overall temporal behavior of the events of a sequence. However, we may find some discrepancies between two sequences having the same form of main-bh. For instance, in Figure 5.2, we obtain two sequences with different sub-bh as “(A before C)” and “(A overlaps C)” though both give the same main-bh as “(A overlaps B), (B overlaps C)” which suggests two different temporal patterns. Hence, we say two sequences have the same (or **identical**) temporal pattern if they share the same main-bh and sub-bh.

Definition 12 [path] *A path exists between two nodes if we found a branch coming from a node to another one directly or a sequence of branches starting from a node, via some intermediate nodes and finally link to the target node. The length of a path is the number of branches being visited.* ■

By finding any paths via both main-bh and sub-bh in the seq-tree, we can obtain the set of temporal patterns. For instance, using the seq-tree shown in Figure 5.1, a path $\langle A \xrightarrow{\text{before}} C \xrightarrow{\text{during}} D \rangle$ is found and we obtain the corresponding pattern “(A before C) & (C during D)”. Thus each path corresponds to a pattern we obtain from a sequence. For the seq-tree, we need to ensure each valid path corresponds to a temporal pattern. Let us consider an example shown in Figure 5.3(a), two sequences have the common partial main-bh as “(A overlaps B), (B before C)” which we can merge them into one path. However, we would come across an invalid path obtained from the tree as $\langle A \xrightarrow{\text{overlaps}} C \xrightarrow{\text{overlaps}} D \rangle$. Hence, we need to exclude such case by splitting the node C as shown in Figure 5.3(b). We then can distinguish two paths showing clearly “(A overlaps C) & (C before E)” and “(A before C) & (C overlaps

D)”. Hence, we realize that only sequences having identical patterns can share same paths in the seq-tree.

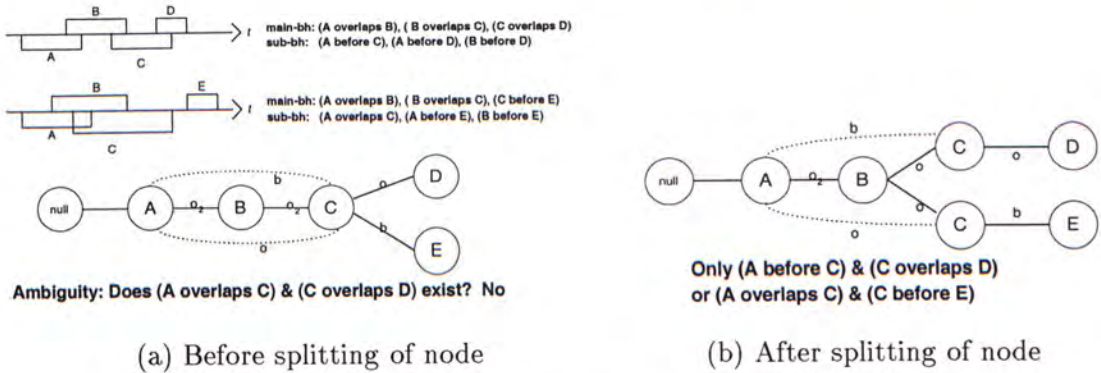


Figure 5.3: Splitting of node C for different sub-bh

With the above observations, a tree-like structure, seq-tree is proposed based on the following considerations.

1. Only L_1 are included as tree nodes so that only potential large patterns are kept.
2. Tree nodes of a path are arranged by the order of events happened in the corresponding sequence. i.e. the end and start time of events. This preserved the relative order of events which is an important information used in the mining process.
3. Each branch between two nodes stores the primitive temporal relation between the events of the binary predicate. Associated with the temporal relation, a *count* is used to store the occurrences of such binary predicate.
4. For each sequence, there is one corresponding path of the tree constructed. If several sequences share an identical temporal pattern, they can merge into one path.
5. Similarly, in case, if two sequences share a common prefix pattern, the shared parts can be merged using one prefix structure. Hence, we can encode the temporal pattern into a compact form with all the information being stored in the seq-tree for further mining process.

We here introduce the design and construction of seq-tree. We consider an example of the database shown in Table 4.1 for illustration. Suppose the minimum support is 33% and the window-size is set to be 30 time units.

item	count
A	6
B	4
C	3
D	3
E	3
F	3
G	3

Table 5.1: Large 1-items

5.4.2 Construction of seq-tree

First, we scan the database to derive a list of large 1-items as shown in Table 5.1. Then, we transform the database into sequences in which we group the events which belonged to the same person as a sequence with only large 1-items, L_1 , are included. For each sequence, we find all possible binary predicates formed by L_1 exist in the sequence and divide them into two groups. Each sequence consists of two kinds of branches: main-bh and sub-bh where the construction of seq-tree is mainly based on the main-bh.

We generate the binary predicates of main-bh by considering consecutive large 1-item as a pair. Since we sort the records with end time of the events, we ensure the previous event stops as late as the following one. In other words, the longest pattern can be formed by linking all the consecutive large 1-item into a sequence as main-bh. By considering other temporal relations between any two events, we form a set of binary predicates as the sub-bh. The binary predicates formed in each sequence are listed as shown in Table 5.2.

Secondly, we use the table of the binary predicates obtained to construct the seq-tree. We start from creating the root labeled with “null”. Look up from the transformed database, for the first sequence, we construct the first path of the tree from main-bh as: $\langle A \xrightarrow{\text{overlaps:1}} B \xrightarrow{\text{before:1}} C \xrightarrow{\text{during:1}} E \xrightarrow{\text{finishes:1}} F \xrightarrow{\text{meets:1}} B \rangle$. Note that the event type is stored as nodes with the former event taken as the parent node and the following event being the child node. Each branch stores the temporal relation and the corresponding count between the pair of nodes is indicated after “:”. As we notice that, starting from the third node onwards along the path, we obtain sub-bh for the node with the preceding nodes. Hence by looking up the table of binary predicates, we add the corresponding sub-bh between the nodes as shown in Figure

person-id	sequence form	
1	main-bh:	(A o B), (B b C), (C d E), (E f F), (F m B)
	sub-bh:	(A b C), (A b E), (B b E), (A b F), (B b F), (C d F), (A b B), (B b B), (C b B), (E m B)
2	main-bh:	(A o B), (B b C), (C m E), (E f F), (F b G)
	sub-bh:	(A b C), (A b E), (B b E), (A b F), (B b F), (C o F), (A b G), (B b G), (C b G), (E b G)
3	main-bh:	(A s D)
	sub-bh:	nil
4	main-bh:	(D b A), (A d G)
	sub-bh:	(D b G)
5	main-bh:	(D b A), (A d G)
	sub-bh:	(D m G)
6	main-bh:	(A o B), (B b C), (C d E), (E f F)
	sub-bh:	(A b C), (A b E), (B b E), (A b F), (B b F), (C d F)

Table 5.2: Transform the database as sequence form

5.4 with small dotted lines.

For the second sequence, since its main-bh shares a common prefix $\langle A \xrightarrow{\text{overlaps:1}} B \xrightarrow{\text{before:1}} C \rangle$ with the existing path of the tree and the corresponding sub-bh $A \xrightarrow{\text{before:1}} C$ is the same, the count of each branch along the prefix is incremented by 1 until reaching the node C. A new branch is created starting from node E as $\langle C \xrightarrow{\text{meets:1}} E \xrightarrow{\text{finishes:1}} F \xrightarrow{\text{before:1}} G \rangle$. We observe that though the nodes of the second sequence excluding the last node are the same as that of the first sequence, different temporal relations in main-bh would lead to new branches.

Regarding the third sequence, the sequence shares only the common node A and we have a new branch for $A \xrightarrow{\text{starts:1}} D$. As there is only one binary predicate for the sequence, no sub-bh is added. The scan of the fourth sequence leads to the construction of the second branch of the tree, $D \xrightarrow{\text{before:1}} A \xrightarrow{\text{during:1}} G$. Again, we add the corresponding sub-bh. For the fifth sequence, we observed that it shares a common main-bh as $D \xrightarrow{\text{before:1}} A \xrightarrow{\text{during:1}} G$. However, different sub-bh $D \xrightarrow{\text{meets:1}} G$ is found with the existing path. Hence, a new branch from A to G is added instead of merging the two sequences together. For the last sequence, since its event types and corresponding relations for both main-bh and sub-bh is identical to that of the first sequence excluding the last event, the prefix path is shared with the count of each relation along the path being incremented by 1.

To facilitate tree traversal, a L_1 look-up table is built in which each event points to its occurrence in the tree via a head of node-link. Nodes with the same event type are linked via such node-links as shown with dotted arrows. After scanning all the sequences, the tree with associated node-links is shown in Figure 5.4. The example leads to the following definition of a sequence tree.

Definition 13 [seq-tree] *A sequence tree (or seq-tree) is a tree-like structure defined below.*

1. *It consists of one root labeled as “null”, a set of event prefix subtrees as the children of the root, and L_1 look-up table.*
2. *Each node in the event prefix subtree consists of event type, node-link, and a list of branches, where event type registers which event this node represents, node-link links to the next node in the seq-tree which has the same event type, or null if there is none. The list of branches link the current node to other nodes along the same path. Each branch stores the temporal relations between two connected nodes and the corresponding count which registers the number of*

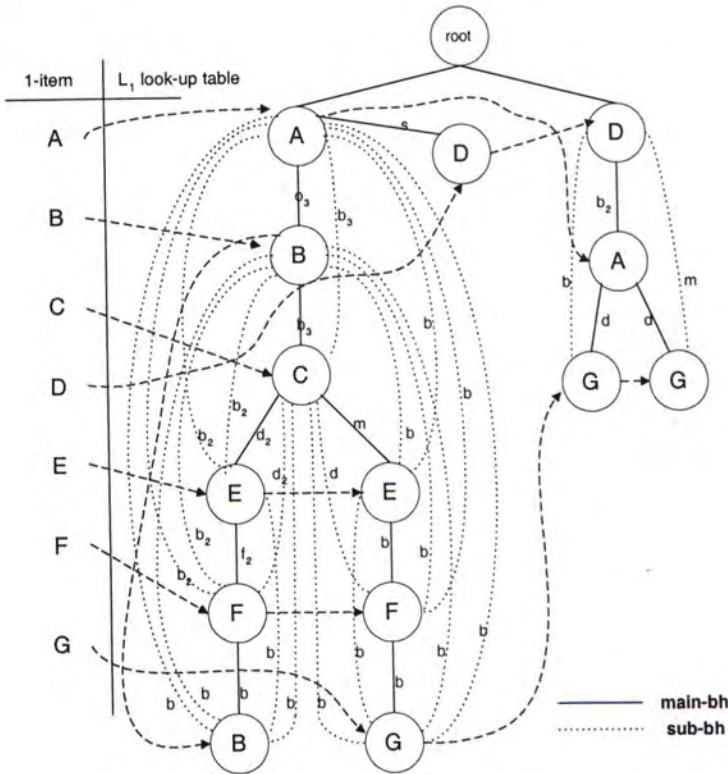


Figure 5.4: The seq-tree constructed using the given example

occurrences of binary predicate represented by the two connected node associated with a temporal relation.

3. Each entry in the L_1 look-up table consists of two fields, (1) event type and (2) head of node-link, which points to the first node in the seq-tree carrying the same event type.

■

Based on this definition, we obtain an algorithm for the seq-tree construction as follows. We scan the database twice. In the first pass, it determines the set of large 1-items which would form large k -items in the mining process. In the next scan, we transform the original database to sequence form with the sorted order of the end time and start time of the events. We find all the binary predicates and group them into two categories as main-bh and sub-bh. Then, we build the corresponding seq-tree based on the above data structure defined. The construction algorithm is shown in Figure 5.5. The construction process is mainly divided into the following steps:

Large 1-items: During the first pass, we determines the large 1-item which any frequent temporal patterns are formed by this set of large items.

Input: A temporal database D and minimum support min_sup

Output: A sequence tree, seq-tree

Algorithm 5.1 seq-tree construction

```

1   $L_1 = \{1\text{-large items}\}$  //tuples containing items with  $min\_sup$ 
2  for each sequence,  $S_k$  do
3      gen_main_branch( $S_k$ ) //by considering consecutive large 1-item as a pair
4      gen_subsidary_branch( $S_k$ ) //by examining other temporal relations between any two events
5  end
6  Create the root of a seq-tree,  $T$ , labeled as "null"
7  for each sequence,  $S_k$  do
8      for each node along main-bh do
9          build_main_branch( $S_k$ ,  $T$ )
10         add_subsidary_branch( $S_k$ ,  $T$ )
11     end
12 end
13 Add node-link to the nodes with the same event type

```

Figure 5.5: Construction of sequence tree, seq-tree

Generate sequence form: In the next pass, we group the records into sequences for different persons. We form the main-bh by examining the temporal relations between two consecutive events along the sequence. Then we find any temporal relations between any two events as sub-bh. Both main-bh and sub-bh of the sequences for the given example is summarized in Table 5.2.

Build main branch: Based on the main-bh and sub-bh, we collect the essential information and construct the corresponding seq-tree. We first use main-bh to add branches to the tree. We insert nodes from the root by first examine any common prefix, including identical sub-bh, if not, create new node and branch which links the new nodes with corresponding relations of the binary predicate. If any sequence shares the same main-bh and same sub-bh stored in the seq-tree, we simply increment the corresponding relation counts by 1. The algorithm for building main branches is summarized in Figure 5.6.

Add subsidiary branch: Upon each sequence, besides building the tree with the main-bh, we need the sub-bh to store other temporal relations between every pair of events along the sequence. We add both main-bh and sub-bh together for each node. After adding the main-bh for a node, we add the corresponding sub-bh by looking up the table storing the binary predicates. Notice that sub-bh

Input : A sequence S_k , root of the seq-tree, T

Output: The seq-tree with the added path of the corresponding sequence S_k

Algorithm 5.2 build_main_branch(S_k, T)

```

1  for the pair (F,rel,E) //(F rel E) is a binary predicate in the main-bh of  $S_k$ 
2      if T has a child N such that N.event=F
3          if N has a child M such that M.event=E and M.rel=rel
4              increment M.rel.count by 1
5          else
6              create a new node M, let M.event=E and M.rel=rel
7              add a branch N and N to M
8          end
9      else
10         create a new node N, let N.event=F
11         create a new node M, let M.event=E and M.rel=rel
12         add a branch from T to N and a branch from N to M
13     end
14 end

```

Figure 5.6: Building main branch

exists only from the third nodes onwards along main-bh with preceding nodes. We add the sub-bh between two nodes as shown in Figure 5.4 with dotted lines. In case, we have sequences having the same main-bh but different sub-bh, splitting of nodes is needed. The algorithm for adding subsidiary branches is shown in Figure 5.7.

Lemma 5 *Given a sequence database D and user-defined minimum support min_sup , its corresponding seq-tree contains the complete information of D for mining LinkSeq.*

Proof: As we observe from the construction process, each sequence in the database is mapped to one path in the seq-tree. The order of events is preserved as we insert each new sequence from the root and follow the order stored in main-bh. For each sequence, all temporal relations for the binary predicates are stored in the seq-tree by main-bh and sub-bh. Hence, all possible temporal relations between events along the sequence are stored. The associated support count records the number of sequences which shares the same set of temporal pattern. Thus, the seq-tree contains the complete information of the database in relevance to mining LinkSeq. ■

Lemma 6 *Without considering the root, the height of the seq-tree is bounded by the*

Input: A sequence S_k , root of the seq-tree, T

Output: The seq-tree with added subsidiary branches of the sequence S_k

Algorithm 5.3 add_subsidary_branches(S_k, T)

```

1  for the pair (F,rel,E) //(F rel E) is a binary predicate in the sub-bh of  $S_k$ 
2      if T has a child N such that N.event=F
3          From the corresponding main-bh to a node M such that M.event=E
4          if(M.sub.event(p)=F not exist) //new sub-bh is added
5              M.sub.event(p) = F
6              M.sub.rel(p) = rel
7              add a branch from N to M as sub-bh
8          else
9              if(M.sub.rel(p)=rel) //same sub-bh with existing branch
10                 increment M.sub.rel(p).count by 1
11             else //same main-bh but different sub-bh
12                 create a new node  $M'$ 
13                 create a new path from N to  $M'$  //separate into two different branches and adjust
                    corresponding counters
14                 mark the node as a splitting node //having same main-bh but different sub-bh
15             end
16         end
17     end
18 end

```

Figure 5.7: Adding subsidiary branches

longest temporal pattern of the sequences in the database, and the size of the tree is bounded by the number of sequences.

Proof: As we can see the maximum number of nodes along a path is determined by the maximum number of events obtained in the corresponding sequence. Hence the height of the tree is bounded by the longest temporal pattern in any sequence in the database. Regarding the size of the seq-tree, as we can see the complex structure of the main-bh and sub-bh, as in case if any sequence having same main-bh but different sub-bh, splitting of nodes is needed. The size of the tree would be quite large. However, the chances of having same main-bh but different sub-bh is not that high, we expect more sequences share the same path in the tree and the size of the tree is smaller than the original database. Also, the number of main-bh in seq-tree cannot be more than that of the number of distinct frequent sequences in the database. Hence the size of the seq-tree is bounded by the number of sequences and the height is bounded by the longest pattern. ■

In fact, the longest pattern obtained is determined by the window-size specified by users. Hence, the height of the seq-tree is also bounded by it. As the paths of the tree is constructed based on main-bh, the window-size here refers to time interval between the first binary predicate and the last binary predicate formed in main-bh. Thus some of the binary predicates of the subsequence or sub-bh are excluded if the longest pattern formed by main-bh exceeds the limit. This leads to different mining results when we compare the performance of the previous methods. Hence in the following sections, we assume the window-size is set to the maximum value that includes all the binary predicates of the sequences.

5.4.3 Mining LinkSeq using seq-tree

In this section, we study how to explore the information stored in the seq-tree for mining the complete set of frequent temporal patterns. We observe some interesting properties of the seq-tree structure which facilitates the mining process.

Property 1 *For any large 1-item, a_i , all possible temporal pattern that contains a_i can be obtained by following a_i 's node-links, starting from the a_i 's in the L_1 look-up table.*

This property is based directly on the construction process of seq-tree that any nodes with the same event type is linked by node-links. This facilitates the extraction of valid path by traversing the seq-tree once following a_i 's node-links.

Property 2 *For any node, there is only one branch connecting two nodes.*

For any sequence, main-bh stores the temporal relations with the preserved order of events. The temporal relations are unique of consecutive events for a sequence. Hence no two branches exist between two nodes of main-bh. Similarly, the temporal relations kept are unique for any two events for sub-bh in a sequence. On the other hand, during the construction process, only identical pattern shares the same path along the tree. Any discrepancy of sub-bh between two sequences having the same main-bh would lead to splitting of nodes. Hence, no two different branches would connect two nodes. In other words, when we found a valid path from a node to another node, we obtain a specific temporal pattern.

Property 3 *To calculate the frequent temporal patterns for a node a_i in a path, only the prefix sub-path of node a_i in the path need to be accumulated, and the support count of every relation in the prefix path should carry the same count as the last branch connected with a_i .*

Let the nodes along the path with event types a_1, \dots, a_n in such an order that a_1 is the root of the prefix sub-tree, a_n is the leaf of the subtree in the path, and a_i ($1 \leq i \leq n$) is the node being referenced. Based on the construction process, for each prefix node a_k ($1 \leq k < i$), and branches, they occur together with a_k exactly $a_i.rel.count$ times. Thus every such prefix node should carry the same count as the last branch connected to a_i . A postfix node a_m (for $i < m \leq n$) along the same path co-occurs with node a_i . However, the patterns with a_m will be generated at the examination of the postfix node a_m , enclosing them will lead to redundant generation of the patterns that would have been generated for a_m . Therefore, we only need to examine the prefix sub-path of a_i in the path.

Based on the constructed seq-tree, we here outline the mining method LinkTree. Without generating large number of candidates, we explore the information stored in the seq-tree as follows. As in the construction of the seq-tree, all the nodes along the same path belong to the same sequence with each pair having one parent node and child node showing the order. Thus the ordering information is kept and the $\&$ is induced among all the pairs of nodes along the same path.

Using the seq-tree of the given example, we start from the last event of the L_1 look-up table, i.e. event type G. Actually, the order of event in the L_1 look-up table is arbitrarily assigned. Note that the mining process can be performed independently for each a_i starting from the look-up table as only prefix sub-path of node a_i are included

when mining for a_i . Other paths of the tree are ignored. Here, for simplicity, we start with node G. According to the paths leading to node G, three paths are found: $\{\langle A \xrightarrow{\text{overlaps:3}} B \xrightarrow{\text{before:3}} C \xrightarrow{\text{meets:1}} E \xrightarrow{\text{finishes:1}} F \xrightarrow{\text{before:1}} G \rangle, \langle D \xrightarrow{\text{before:2}} A \xrightarrow{\text{during:1}} G \rangle, \langle D \xrightarrow{\text{before:2}} A \xrightarrow{\text{during:1}} G \rangle$ with different sub-bh} By examining the counter values of all the branches connecting node G, we have the following L_k containing G. We first consider all the intermediate nodes connecting node G via main-bh and sub-bh. For L_2 , we consider any binary predicates having enough support count. From the first path containing node G, we find from both main-bh and sub-bh and obtain “(A before G):1”, “(B before G):1”, “(C before G):1”, “(E before G):1” and “(F before G):1”. However, all of them do not have enough support. For the following two paths, only “(A during G):2” has enough support. Hence, we have “(A during D)” as L_2 . For L_3 , we look for any path connecting to node G with length being 2, reaching 3 nodes along the path. This time, we can start with the intermediate nodes which are included in L_2 previously found and we have “(D before A) & (A during G)” by traversing the main-bh of the last two paths. Since we find no more path having length greater than 2 with enough support count, the search for L_4 for node G terminates.

For node with event type F, we only focus on the prefix along the selected path to avoid any repetition of patterns generated. By considering any path that reaches node F, we obtain two prefix paths $\{\langle A \xrightarrow{\text{overlaps:3}} B \xrightarrow{\text{before:3}} C \xrightarrow{\text{during:2}} E \xrightarrow{\text{finishes:2}} F \rangle, \langle A \xrightarrow{\text{overlaps:3}} B \xrightarrow{\text{before:3}} C \xrightarrow{\text{meets:1}} E \xrightarrow{\text{finishes:1}} F \rangle\}$. Considering the binary predicates in both main-bh and sub-bh, we have L_2 as “(A before F):3”, “(B before F):3”, “(C during F):3” and “(E finishes F):3”. For L_3 , again we search for any path having length being 2 by starting with the L_2 found previously and we have “((A before E) & (E finishes F)):3”, “((A before C) & (C during F)):2”, “((A overlaps B) & (B before F)):3”, “((B before C) & (C during F)):2”, “((B before E) & (E finishes F)):3”, “((C during E) & (E finishes F)):2”. Then we proceed to find L_4 by finding path having length being 3 and we have “((A overlaps B) & (B before C) & (C during F)):2”, “((A overlaps B) & (B before E) & (E finishes F)):3”, “((A before C) & (C during E) & (E finishes F)):2”, “((B before C) & (C during E) & (E finishes F)):2”. For L_5 , we have the longest pattern “((A overlaps B) & (B before C) & (C during E) & (E finishes F)):2”. As we cannot find any path with length longer than 4, the search for large sequence associated with F terminates.

Consider nodes with event type E, we derive two prefix paths $\{\langle A \xrightarrow{\text{overlaps:3}} B \xrightarrow{\text{before:3}} C \xrightarrow{\text{during:2}} E \rangle, \langle A \xrightarrow{\text{overlaps:3}} B \xrightarrow{\text{before:3}} C \xrightarrow{\text{meets:1}} E \rangle\}$. Similarly, by first finding the binary predicates as L_2 , we obtain “(A before E):3”, “(A before E):3”, “(C during E):3”. For L_3 , we have “((A overlaps B) & (B before E)):3”, “((A before C) & (C during

node	LinkSeq
G	(A during G):2, ((D before A) & (A during G)):2
F	(A before F):3, (B before F):3, (C during F):2, (E finishes F):3, ((A before E) & (E finishes F)):3, ((A before C) & (C during F)):2, ((A overlaps B) & (B before F)):3, ((B before C) & (C during F)):2, ((B before E) & (E finishes F)):3, ((C during E) & (E finishes F)):2, ((A overlaps B) & (B before C) & (C during F)):2, ((A overlaps B) & (B before E) & (E finishes F)):3, ((A before C) & (C during E) & (E finishes F)):2, ((B before C) & (C during E) & (E finishes F)):2, ((A overlaps B) & (B before C) & (C during E) & (E finishes F)):2
E	(A before E):3, (B before E):3, (C during E):2, ((A overlaps B) & (B before E)):3, ((A before C) & (C during E)):2, ((B before C) & (C during E)):2, ((A overlaps B) & (B before C) & (C during E)):2
D	\emptyset
C	(A before C):3, (B before C):3, ((A overlaps B) & (B before C)):3
B	(A overlaps B):3
A	(D before A):2

Table 5.3: Mining LinkSeq by traversing *seq_tree*

E)):2", "((B before C) & (C during E)):2". The longest pattern gives L_4 as "((A overlaps B) & (B before C) & (C during E)):2" and terminates the search.

For nodes with event type D, we only find a path with length being 2 as $\langle A \xrightarrow{\text{starts:1}} D \rangle$. However, it does not have enough support. Hence no L_k is found and the search for node D terminates.

For other nodes with event type C, B and A, same mining mechanism is used as finding any path from the seq-tree as L_k . The LinkSeq generated from each node are summarized in Table 5.3.

The mining process mainly focuses in the discovery of any valid path starting from an event in the L_1 look-up table to other preceding nodes along the path of the seq-tree with different path lengths. The valid paths with enough support count obtained is used for the generation of temporal patterns. The algorithm for mining temporal pattern from seq-tree is shown in Figure 5.8. It can be summarized by the

following steps:

Look-up table: Start from any event of the L_1 look-up table. The order of events in the look-up table is arbitrarily assigned. As we can find each corresponding pattern associate with the event separately, the order of events of which we start with the mining process is independent with the results obtained.

Path extraction: After determining the associated event for the temporal pattern, we extract any path of the seq-tree that leads to the associated event for further support counting. Only prefix of the path is considered.

Valid path: By finding any valid path that links the associated event to other preceding nodes with $k - 1$ length, we generate potential k -item. By further examination of the counter values, those having enough support count become large k -item in result.

Pattern generation: From the k -item obtained, we generate the corresponding temporal pattern, LinkSeq, easily as one k -item corresponds to one temporal pattern.

During the search of any valid paths leading to the ending node a_i along the prefix path, we start from any neighboring nodes with length of path being 1 to form L_2 . We then form L_3 by starting from the corresponding nodes which previously found in L_2 , say b_i . Again we find any neighboring nodes of b_i and form L_3 with path length being 2. The intuition is that any nodes connecting b_i would end up with a_i also. This is obvious as when we can find a valid path from a neighboring node, say c_i , reaching b_i as an intermediate node, would reach a_i through the branch previous found in L_2 . Thus the mining process is performed recursively along the prefix paths found to generate all L_k .

Lemma 7 *Algorithm 5.4 finds all the potentially large k -items by traversing the tree and examining any valid paths.*

Proof With lemma 5, we are certain that all the binary predicates along the sequences are kept in the tree. As we find all binary predicates between events along the sequences and the ordered of events are preserved in the paths of the tree, any possible temporal pattern of the sequence can be found by finding any valid path linking the nodes. Also only identical pattern shares the same path, all the events stored along the path belong to the same sequence. With property 1, any possible temporal pattern

Input: A seq-tree T , minimum support min_sup

Output: The set of frequent temporal patterns

Algorithm 5.4 Mining frequent temporal patterns with seq-tree

```

1  if  $T$  has only one branch
2      return all valid paths generated from the branch for each node in that branch with  $rel.count \geq min\_sup$ 
3  else
4      for each node with event type  $e_i$  in the  $L_1$  look-up table
5          extract all prefix paths associates with  $e_i$ 
6          for each  $k \geq 2$ 
7              if there exist a path from node with event type  $e_i$  to other preceding nodes of length
                =  $k-1$  with  $rel.count \geq min\_sup$ 
8                  generate corresponding  $L_k$ 
9              end
10         end
11     end
12 end

```

Figure 5.8: Mining frequent temporal pattern from seq-tree

including the associated events can be found. Hence by scanning all the events in the L_1 look-up table and traversing the tree, we generate all large k -items. ■

Lemma 8 *Any large k -item formed from Algorithm 5.4 represents a frequent temporal pattern.*

Proof Based on property 2, we ensure a path starting from any node would only lead to a deterministic node. Hence, no path represents a non-exist temporal pattern in any sequence. In other words, each path we found corresponds to a temporal pattern. By examining the counter values, we generate large k -item which represents a specific frequent temporal pattern. ■

With the properties and lemmas above, we show that the algorithm correctly finds the the complete set of frequent temporal patterns, LinkSeq. Without any candidates generated during the process, we obtain all the L_k by traversing the tree independently for each a_i in L_1 look-up table. For each frequent event a_i , an extracted set of prefix paths is extracted. The mining process is then performed recursively along the prefix paths to generate all L_k . As the seq-tree is usually smaller than the size of the database, the mining process takes less storage than that of the previous methods,

LinkApp and LinkSeq, where large number of candidates are formed.

5.5 Performance Study

We use the same set of synthetic data introduced in the previous chapter. We compare the performance of four methods, including LinkApp, modified LinkApp, LinkTwo and LinkTree. Modified LinkApp is basically same as LinkApp except it generates L_2 without C_2 using the approach developed in Chapter 4, while the other three methods are described in the previous sections. Note that LinkTwo also forms L_2 without C_2 as we found in the previous chapter that, this approach helps in improving the mining process.

We start by studying the effect of minimum support (min_sup) on the processing time. We used 6 values of min_sup as shown in Figure 5.9. The window size (win_size) is set to be 200 time units for the test. The figure shows that the execution times for the four methods decrease when the minimum support increases. As less large items are formed when the support threshold increases, the size of the candidate set in each iteration for the three methods, LinkTwo, LinkApp and modified LinkApp, decreases dramatically. Thus less time is required for support counting. On the other hand, for LinkTree, as less L_k are formed during the process, the time for searching valid path decreases for greater support threshold. We observe that LinkTree outperforms the other three methods, especially for small support threshold. It is likely that only a slight increase in the execution time for LinkTree. As no generation of candidates is needed, we avoid the high cost of support counting phase for large number of candidates generated during the process. Comparing the other three methods, LinkTwo works better. As for both LinkApp and modified LinkApp, we generate AppSeq first in each iteration before we get LinkSeq. Further computation time is needed to search from the corresponding binary predicates in L_2 , in addition to find L_k .

Table 5.4 shows the number of LinkSeq obtained with different values of min_sup . The number of sequences decreases with increasing value of min_sup .

We then study the scale-up effects which we examine how the performance varies with the number of sequences. The number of sequences is increased ten-fold, ranging from 10K to 100K. We set the $min_sup = 0.001$ for the four methods. Figure 5.10 shows the scalability results. LinkApp grows rapidly when comparing with the other three methods. LinkTwo and modified LinkApp scales linearly in the same manner. As more large items are formed in the first few passes of the mining process, the

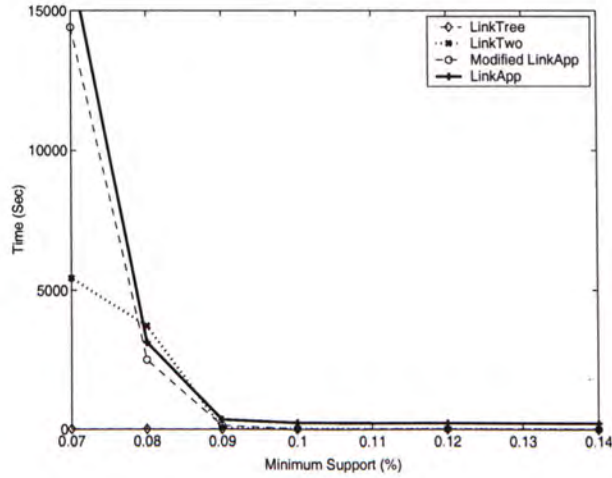


Figure 5.9: Variation on minimum support

min_sup	no. of resulting sequences	max. sequence length
0.0007	31837	10
0.0008	17414	10
0.0009	6022	9
0.0010	2623	9
0.0012	154	6
0.0014	20	4

Table 5.4: Number of LinkSeq obtained with different *min_sup*

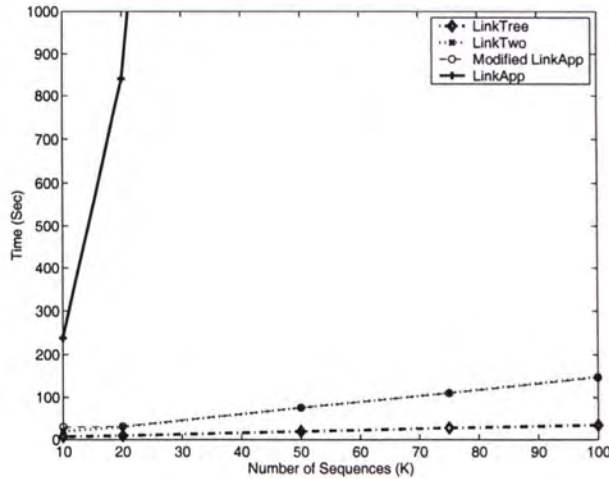


Figure 5.10: Scale-up: Number of sequences

approach of generating L_2 without C_2 did help in generating LinkSeq for LinkTwo and modified LinkApp. On the other hand, LinkTree scales much better than the other three methods. As the number of sequences grows up, the difference between LinkTree and the other three methods becomes larger and larger. This shows the advantage of eliminating the generation of large number of candidates during the process.

We finally studied the scale-up as we increase the average number of events per sequence. The number of sequences used is 10K and kept constant. We vary the average number of events per sequence from 2.5 to 25 and set $min_sup = 0.0025$ for the four methods. Figure 5.11 shows how the methods scale up as the number of events per sequence is increased. Like the case for increasing the number of sequences, the execution time for LinkApp grows dramatically with increasing number of events per sequence while LinkTwo and modified LinkApp scales up linearly in the same manner. As there are tremendous number of L_2 formed with increasing number of events in the sequences, LinkApp suffers from the high cost of C_2 in the first few passes. LinkTwo and modified LinkApp uses the same approach to avoid this problem. Regarding LinkTree, when the number of events per sequence increases, the execution time for LinkTree increases more rapidly than that of LinkTwo and modified LinkApp. This may due to the fact that longer sequences are formed, and thus longer paths along the tree are obtained. Hence longer execution time for searching the valid paths for generating L_k is required.

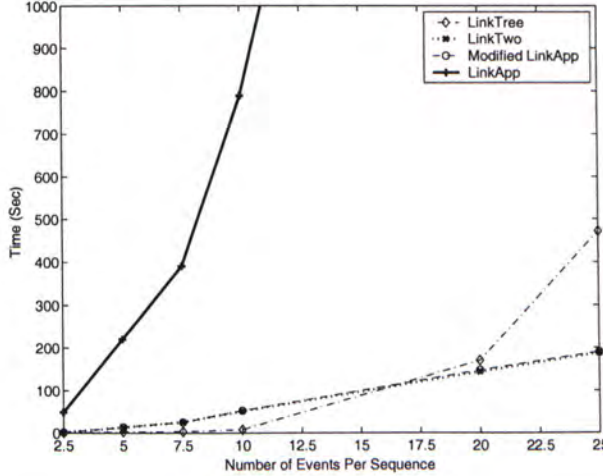


Figure 5.11: Scale-up: Number of events per sequence

5.6 Discussions

As we can see, the use of sequence tree structure facilitates efficient support counting of large items. The LinkSeq is in general simple in nature and only relative order of events are concerned. This information is maintained by the seq-tree formed as we insert all nodes from the root. We then suggest to apply similar mechanism for mining AppSeq. However, when we look into the formation of AppSeq and LinkSeq, we found that AppSeq is more complex in structure and requires the mapping of start time and end time of the composite patterns during the mining process. This involves the storage of pid-list with pid, start and end time of each composite item represented by the node. For example, for the composite item “A overlaps B” with the end time being $[5,12],[12,22],[14,22]$ associated with node B. As a result, more memory is needed to store the pid-list and more processing time is needed for looking up the corresponding temporal relations between the composite pattern and the following atomic pattern along the path. Hence, an overhead of increasing the processing time in the mining makes it not favourable to employ the tree structure for mining AppSeq.

5.7 Summary

In this chapter, we propose several methods for discovering the second temporal pattern, LinkSeq. LinkSeq is found comparatively simpler in structure than that of AppSeq as examination of start time and end time is neglected. Only the orderings of the binary predicates are essential for the mining process. Thus a tree-like structure with a mining algorithm, LinkTree, is proposed. Besides, we suggest to extract

LinkSeq from AppSeq and by modifying the previous method, AppOne, we obtain our second method, LinkTwo. We compare the performance of the methods with a set of experiments. Overall, LinkTree is suitable for mining LinkApp when the minimum support threshold reduces.

Chapter 6

Conclusion and Future Work

6.1 Conclusion

Based on previous studies on temporal data mining and mining in sequence data, we proposed and studied methods for mining temporal patterns for interval-based events. We extend the current work to accommodate temporal interval data which has long been overlooked in the past. Discovery of temporal pattern involving interval data is useful in a number of complex data analysis scenarios. As other than serial and parallel ordering of events, inter-relations such as “overlaps, during”, etc., can be found which enriches the expressive power of the temporal patterns. Interesting relationships among events can be found and thus, give some insight into causal relationships. Two interesting patterns, namely AppSeq and LinkSeq, are suggested to describe the complex relations among events. Both patterns are simple and useful to capture the temporal behavior of the events.

We developed several methods for discovering the two interesting patterns and a set of experiments are used to compare the performance of the methods. Regarding AppSeq, for comparison we consider the mining of a slightly more complex temporal pattern, A2, which is a variate of the original pattern, A1. An Apriori-like approach which an iterative method is used for mining both patterns. We propose to use an item-list format to store the temporal data to facilitate fast computation in the support counting phase. From experiments, we find that the computation time required for the original pattern, A1, is much more acceptable. On the other hand, using the approach of generating binary predicates without generating 2-candidates with the use of a tree-like structure is proved to be useful in improving the efficiency. We can further investigate other data structures as well as algorithms that facilitate efficient mining process.

Another temporal pattern, LinkSeq, forms in a similar way as that of the previous pattern, AppSeq, but is found comparatively simpler in structure. An iterative method, LinkTwo, using item-list is suggested. Besides, a tree-like structure with a mining algorithm, LinkTree, is proposed for finding LinkSeq. It stores crucial information in a compact way as a tree such that only two scanning of the database is required to find all the frequent pattern. Without generating any candidates, we obtain the complete set of temporal patterns by traversing the tree. This approach was shown to be efficient for mining LinkSeq.

In fact, there are tremendous way to express the temporal relations between interval data. We discover that the number of temporal patterns can be prohibitively large and also many of such patterns may be complicated and of little value to the users. Hence we restrict our interest to two types of temporal patterns which are simple and meaningful.

6.2 Future Work

So far, as we have only considered the discovery of frequent or “large” temporal patterns. We may consider the generation of association rules of the form $A \rightarrow_{rel_i} B$ where A and B are some temporal patterns. The meaning of such a rule is that for some temporal relation rel_i , among the support of all frequent patterns of the form $A rel_i B$, the percentage of support for the pattern $A rel_i B$ is sufficiently high. This can give some indication of causal relationship among temporal patterns of events and can be studied.

Contrary to the case of time points, instead of taking the ordering of events as a measure for finding interesting pattern, we may take the length of time interval that interacts between two events as a measure other than only temporal relations. For instance, “A overlaps B” where the overlapping interval lasts for 4 time units. This can be read as “event B overlaps with event A for 4 time units before event A ends”. As more accurate information is extracted, we can further study this approach by taking into consideration of time intervals where the event lasts.

On the other hand, so far we mainly work on two approaches in the discovery of both patterns. One is an Apriori-like approach with the use of item-list to facilitate efficient support counting process. The other is a tree-like structure which is used for mining LinkSeq. In fact, other data structure like frequent item matrix which proposed recently to find sequential pattern [36], may help in our mining problem. We can further investigate this approach in the future.

Bibliography

- [1] R. Agrawal, C. Faloutsos, and A. Swami. Efficient similarity search in sequence database. In *Proceedings of the 4th International Conference on Foundations of Data Organization and Algorithms*, pages 69–84, October 1993.
- [2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 94–105, June 1998.
- [3] R. Agrawal, S. Ghosh, T. Imielinski, B. Iyer, and A. Swami. An interval classifier for database mining applications. In *Proceedings of the 18th International Conference on Very Large Data Bases*, pages 560–573, August 1992.
- [4] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 207–216, May 1993.
- [5] R. Agrawal, K.I. Lin, H.S. Sawhney, and K. Shim. Fast similarity search in the presence of noise, scaling, and translation in time-series databases. In *Proceedings of the 21st International Conference on Very Large Databases*, pages 490–501, September 1995.
- [6] R. Agrawal, G. Psaila, E.L. Wimmers, and M. Zaki. Querying shapes of histories. In *Proceedings of the 21st International Conference on Very Large Databases*, pages 502–514, September 1995.
- [7] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Database*, pages 487–499, September 1994.
- [8] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proceedings of the 11th International Conference on Data Engineering*, pages 3–14, March 1995.

- [9] J.F. Allen. Maintaining knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.
- [10] T. Amagasa, M. Aritsugi, and Y. Kanamori. Implementing time-interval class for managing temporal data. In *Proceedings of the 9th International Workshop on Database and Expert Systems Applications*, pages 843–849, August 1998.
- [11] G. Berger and A. Tuzhilin. Discovering unexpected patterns in temporal data using temporal logic. In *Temporal Databases - Research and Practice*, pages 281–309, 1998.
- [12] C. Bettini, S.Wang, and S. Jajodia. Temporal semantic assumptions and their use in databases. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):277–296, 1998.
- [13] C. Bettini, S. Wang, S. Jajodia, and J.L. Lin. Discovering frequent event patterns with multiple granularities in time sequences. *IEEE Transactions on Knowledge and Data Engineering*, 10(2):222–236, 1998.
- [14] S. Chakrabarti, S. Sarawagi, and B. Dom. Mining surprising patterns using temporal description length. In *Proceedings of the 24th International Conference on Very Large Databases*, pages 606–617, August 1998.
- [15] R. Chandra, A. Segev, and M. Stonebraker. Implementing calendars and temporal rules in next generation databases. In *Proceedings of the 10th International Conference on Data Engineering*, pages 264–273, February 1994.
- [16] X. Chen and I. Petrounias. An architecture for temporal data mining. In *IEE Colloquium on Knowledge Discovery and Data Mining*, pages 8/1–8/4, May 1998.
- [17] X. Chen and I. Petrounias. Language support for temporal data mining. In *Proceedings of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 282–290, September 1998.
- [18] X. Chen and I. Petrounias. Mining temporal features in association rules. In *Proceedings of the 3rd European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 295–300, September 1999.
- [19] X. Chen, I. Petrounias, and H. Heathfield. Discovering temporal association rules in temporal databases. In *International Workshop on Issues and Applications of Database Technology*, pages 312–319, July 1998.

- [20] G. Das, D. Gunopulos, and H. Mannila. Finding similar time series. In *Proceedings of the 1st European Symposium on Principles of Data Mining and Knowledge Discovery*, June 1997.
- [21] G. Das, K.I. Lin, H. Mannila, G. Renganathan, and P. Smyth. Rule discovery from time series. In *Proceedings of the 2nd European Symposium on Principles of Data Mining and Knowledge Discovery*, pages 16–22, September 1998.
- [22] R. Elmasri, G. Wu, and V. Kouramajian. The time index and the monotonic b+-tree. In *Temporal databases: theory, design and implementation, Chapter 18*, 1993.
- [23] R. Elmasri, G.T.J. Wu, and Y.J. Kim. The time index: An access structure for temporal data. In *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 1–12, August 1990.
- [24] M. Ester, H.P. Kriegel, and J. Sander. A density-based algorithm for discovering clusters in large spatial databases. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 226–231, August 1996.
- [25] C. Faloutsos, M. Ranganathan, and Y. Manolopoulos. Fast subsequence matching in time-series databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, June 1994.
- [26] U. Fayyad, D. Haussler, and P. Storoltz. Mining scientific data. *Communications of the ACM*, 29(11):51–57, 1996.
- [27] U.M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy. *Advances in Knowledge Discovery and Data Mining*. AAAI Press/MIT Press, 1996.
- [28] L. Forlizzi, R.H. Güting, E. Nardelli, and M. Schneider. A data model and data structures for moving objects databases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, May 2000.
- [29] N. Friedman, D. Geiger, and M. Goldszmidt. Bayesian network classifiers. *Machine Learning*, 29(2):131–163, 1997.
- [30] A. Galton. *Temporal logics and their applications*. London: Academic Press, 1987.

- [31] M.N. Garofalakis, R. Rastogi, and K. Shim. Spirit:sequential pattern mining with regular expression constraints. In *Proceedings of the 25th International Conference on Very Large Database*, pages 223–234, September 1999.
- [32] S. Guha, R. Rastogi, and K. Shim. Cure: An efficient clustering algorithm for large databases. In *Proceedings of the ACM SIGMOD International conference on Management of Data*, pages 73–84, June 1998.
- [33] V. Guralnik and J. Srivastava. Event detection from time series data. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 33–42, August 1999.
- [34] V. Guralnik, D. Wijesekera, and J. Srivastava. Pattern directed mining of sequence data. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 51–57, 1998.
- [35] J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Proceedings of the 15th International Conference on Data Engineering*, pages 106–115, March 1999.
- [36] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.C. Hsu. Freespan: Frequent pattern-projected sequential pattern mining. In *Proceedings of the 6th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, August 2000.
- [37] J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of the SIGMOD 2000 International Conference on Management of Data*, pages 1–12, May 2000.
- [38] M. Holsheimer, M. Kersten, H. Mannila, and H. Toivonen. A perspective on databases and data mining. In *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining*, pages 150–155, August 1995.
- [39] C.S. Jensen, C.E. Dyreson, and M. Böhlen et al. The consensus glossary of temporal database concepts - february 1998 version. In *Temporal Databases - Research and Practice, Lecture Notes in Computer Science, 1399*, pages 338–366, 1988.
- [40] C.S. Jensen and R.T. Snodgrass. Temporal data management. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):36–43, 1999.

- [41] R. J. Bayardo Jr. and R. Agrawal. Mining the most interesting rules. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 145–154, August 1999.
- [42] D. Keim, H. Kriegel, and T. Seidl. Supporting data mining of large databases by visual feedback queries. In *Proceedings of the 10th International Conference on Data Engineering*, pages 302–313, February 1994.
- [43] M. Klemettinen, H. Mannila, P. Ronkainen, H. Toivonen, and A. I. Verkamo. Finding interesting rules from large sets of discovered association rules. In *Proceedings of the 3rd International Conference on Information and Knowledge Management*, pages 401–407, November/December 1994.
- [44] N. Kline. An update of the temporal database bibliography. *SIGMOD Record*, 22(4):66–80, 1993.
- [45] K. Koperski, J. Han, and N. Stefanovic. An efficient two-step method for classification of spatial data. In *Proceedings of the International Symposium on Spatial Data Handling*, July 1998.
- [46] F. Korn, A. Labrinidis, Y. Kotidis, and C. Faloutsos. Ratio rules: A new paradigm for fast, quantifiable data mining. In *Proceedings of the 24th International Conference on Very Large Databases*, pages 582–593, August 1998.
- [47] P.J. Rousseeuw L. Kaufman. *Finding groups in data : an introduction to cluster analysis*. John Wiley & Sons, 1990.
- [48] B. Leban, D.D. McDonald, and D.R. Forster. A representation for collections of temporal intervals. In *Proceedings of the 13th National Conference on Artificial Intelligence*, pages 367–371, August 1996.
- [49] H. Mannila. Data mining: machine learning, statistics, and databases. In *Proceedings of the 8th International Conference on Scientific and Statistical Database Management*, pages 18–20, June 1996.
- [50] H. Mannila and H. Toivonen. Discovering generalized episodes using minimal occurrences. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 146–151, August 1996.
- [51] H. Mannila, H. Toivonen, and A. Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the 1st International Conference on Knowledge Discovery and Data Mining*, pages 210–215, August 1995.

- [52] L.E. McKenzie. Bibliography: temporal databases. *SIGMOD Record*, 15(4):40–52, 1986.
- [53] A. Montanari and B. Pernici. Temporal reasoning. In *Temporal Databases: Theory, Design and Implementation*, pages 534–562, 1993.
- [54] R.T. Ng and J. Han. Efficient and effective clustering methods for spatial data mining. In *Proceedings of the 20th International Conference on Very Large Database*, pages 144–155, September 1994.
- [55] B. Özden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proceedings of the 14th International Conference on Data Engineering*, pages 412–421, February 1998.
- [56] B. Padmanabhan and A. Tuzhilin. Pattern discovery in temporal databases: A temporal logic approach. In *Proceedings of the 2nd International Conference on Knowledge Discovery and Data Mining*, pages 351–354, August 1996.
- [57] J.S. Park, M.S. Chen, and P.S. Yu. An effective hash-based algorithm for mining association rules. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 175–186, June 1995.
- [58] J. Pei, J. Han, B. Mortazavi-asl, and H. Zhu. Mining access patterns efficiently from web logs. In *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, April 2000.
- [59] G. Piatetsky-Shapiro and W.J. Frawley. *Knowledge Discovery in Databases*. AAAI Press/MIT Press, 1991.
- [60] J.R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.
- [61] A. Swami R. Agrawal, T. Imielinski. Database mining: A performance perspective. *IEEE Transactions on Knowledge and Data Engineering*, 5(6):914–925, 1993.
- [62] C.P. Rainsford. *Accommodating Temporal Semantics in Data Mining and Knowledge Discovery*. PhD thesis, Computer and Information Science, University of South Australia, November 1999.
- [63] C.P. Rainsford and J.F. Roddick. The attribute-oriented induction of rules from temporal interval data. In *Proceedings of the 8th International Database Workshop*, pages 108–118, July 1997.

- [64] C.P. Rainsford and J.F. Roddick. Adding temporal semantics to association rules. In *Proceedings of the 3rd European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 504–509, September 1999.
- [65] S. Ramaswamy, S. Mahajan, and A. Silberschatz. On the discovery of interesting patterns in association rules. In *Proceedings of the 24th International Conference on Very large Databases*, pages 368–379, August 1998.
- [66] J.F. Roddick and M. Spiliopoulou. A bibliography of temporal, spatial, and spatio-temporal data mining research. In *Proceedings of the 5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 34–38, August 1999.
- [67] John F. Roddick and Myra Spiliopoulou. Temporal data mining: Survey and issues. *IEEE Transactions on Knowledge and Data Engineering*, 11(1):36–43, 2000.
- [68] B. Salzberg and V.J. Tsotras. Comparison of access methods for time-evolving data. *ACM Computing Surveys*, 31(2):158–221, 1999.
- [69] M. Saraee and C. Theodoulidis. Knowledge discovery in temporal databases. In *Proceedings of IEE Colloquium on Knowledge Discovery in Databases*, pages 1–4, February 1995.
- [70] P.G. Selfridge, D. Srivastava, and L.O. Wilson. Idea: Interactive data exploration and analysis. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 24–34, June 1996.
- [71] P. Shenoy, J.R. Haritsa, S. Sudarshan, G. Bhalotia, M. Bawa, and D. Shah. Turbo-charging vertical mining of large datbases. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 22–33, May 2000.
- [72] R.T. Snodgrass and A. Gad et al A. Ilsoo. *The TSQL2 Temporal Query Language*. Kluwer Academic Publishers, 1995.
- [73] M. Spiliopoulou and L.C. Faulstich. Wum: A tool for www utilization analysis. In *Proceedings of the International Workshop on the Web and Databases, WebDB'98*, March 1998.

- [74] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 1–12, June 1996.
- [75] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proceedings of the 5th International Conference on Extending Database Technology*, pages 3–17, March 1996.
- [76] R.B. Stam and R. Snodgrass. A bibliography on temporal databases. *Data Engineering*, 7(4):53–61, 1988.
- [77] A.U. Tansel. Temporal relational data model. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):464–479, 1997.
- [78] A.U. Tansel, J. Clifford, S. Gadia, S. Jajodia, A. Segev, and R. Snodgrass. *Temporal Databases: Theory, Design and Implementation*. Benjamin/Cummings, 1993.
- [79] A.U. Tansel and E. Tin. The expressive power of temporal relational query languages. *IEEE Transactions on Knowledge and Data Engineering*, 9(1):120–134, 1995.
- [80] D. Toman. Point vs. interval-based query languages for temporal databases. In *Proceedings of the 5th ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 58–67, June 1996.
- [81] A. Tuzhilin and J. Clifford. A temporal relational algebra as a basis for temporal relational completeness. In *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 13–23, August 1990.
- [82] G.M. Weiss and H. Hirsh. Learning to predict rare events in event sequences. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 359–363, August 1998.
- [83] Y. Wu, S. Jajodia, and X.S. Wang. Temporal database bibliography update. In *Temporal Databases - Research and Practice, Lecture Notes in Computer Science*, pages 338–366, 1998.
- [84] X. Ye and J. A. Keane. Mining association rules in temporal databases. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 2803–2808, Oct 1998.

- [85] M.J. Zaki. Efficient enumeration of frequent sequences. In *Proceedings of the 7th International Conference on Information and Knowledge Management*, pages 68–75, November 1998.
- [86] M.J. Zaki, N. Lesh, and M. Ogihara. Planmine: Sequence mining for plan failures. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining*, pages 369–373, August 1998.
- [87] M.J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining*, pages 283–286, August 1997.
- [88] T. Zhang, R. Ramakrishnan, and M. Livny. Birch: An efficient data clustering method for very large databases. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 103–114, June 1996.

CUHK Libraries



003803515