TEMPORAL DATA MINING: ALGORITHMS FOR TEMPORAL ASSOCIATION RULES

A thesis submitted to The University of Manchester for the degree of
Doctor of Philosophy
in the Faculty of Humanities

2006

Xiao Fu
School of Informatics

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

(EPNJG)

List of Contents

List of Figures and Tables

Abstract

Studies on data mining are being pursued in many different research areas, such as Machine Learning, Statistics, and Databases. The problem of association rules, motivated by the application of the market basket analysis, forms one of the most important research areas of the database perspective of data mining. This thesis considers the unsolved challenge of temporal association rule mining techniques, that being to directly discover temporal association rules, through two fields.

The focus of the study is on directly searching interval association rules from temporal transaction datasets, which is the main problem of directly mining temporal association rules. Some special issues about interval association rules are highlighted together with the definitions relating to interval association rule mining techniques. The method introduced for mining interval association rules applies the special techniques for balancing the two aspects of the interval association rule mining problem, which are the searching of longest intervals and the searching of association rules. After the algorithm description and implementation, the efficiency and effectiveness of this interval association rule mining method are evaluated by using both synthetic and real business datasets.

The second focus of this thesis is a tree projection algorithm, which will accelerate most itemset-based temporal association rule mining methods. This delivers the idea of projecting the temporal transaction dataset into a structured temporal pattern base to accelerate the temporal association rule mining process. The tree structure is highlighted for this purpose by introducing the special requirements of its use. This tree projection method involves projecting the temporal transaction dataset into the tree and the method of mining temporal association rules from the projected tree. After the discussion on implementation, which presents the key programming structures of this method, the evaluation exercise uses a real business sales dataset to evaluate the efficiency and effectiveness of the technique.

Declaration

No portion of the work referred to in the thesis has been submitted in support of an application for another degree or qualification of this or any other university or other institute of learning

Copyright Statement

Acknowledgements

There are many people who have contributed to this study. First of all, I wish to offer my special thanks to my supervisor Dr Ilias Petrounias, who has provided very helpful information and guidance throughout the whole process of this research. Then, I would like to express my gratitude to the School of Informatics in the Faculty of Humanities of the University of Manchester (formerly the Department of Computation of UMIST) for a range of support, which is certainly vital to the completion of this thesis.

I would like to give the most faithful thanks to my mother, SuXia Xue, and my fiancée AoFeng Mu, for their continued financial and emotional support during all these years. Without their patience, tolerance, and belief in me, the completion of this thesis would not have been possible.

Particular thanks are also given to Mrs. Yi Qian, mother to my friend JunTao Luo, who provided me with the precious evaluation data for my research.

It is impossible to list all those people who have contributed to this thesis, and all the sources of energy I have received, but I would especially like to acknowledge the sense of balance, pleasure and happiness provided by my two cats, Bora and Soda, who have unfailingly managed to lighten my spirits during this long research journey.

1 Introduction

1.1 Background

1.1.1 Data Mining

Coupled with the newly-improved data storage and management technologies, the advances in business, scientific and medical data collections make it quicker and easier to collect and store data. The explosive growth in databases has provided huge amounts of data that may contain much important and useful information (knowledge). This enormity of information is likely to be explored by many organisations or individuals for making crucial business decisions, and/or for improving their existing understanding of business behaviour, or medical and scientific phenomena. Nonetheless, despite their potential, large databases can become 'data tombs' [Han and Kamber, 2001], because human analysts do not have the tools to extract meaningful information from such huge amounts of data. In order to deal with the analysis of data in large databases, Data Mining technology is needed.

Data mining is considered to be a step in the Knowledge Discovery in Databases (KDD) process, which includes other phases such as data pre-processing and pattern evaluation [Fayyad et al., 1996]. These steps are necessary in order to ensure that the extracted patterns are useful. Although some components of data mining technology have been undergoing development for many years in research areas such as statistics,

artificial intelligence, and machine learning, applying these techniques to large databases is still a challenging problem [Chen, 1999]. The important distinguishing characteristic of data mining is that the volume of data is assumed to be very large. Therefore, the running time of a data mining algorithm must be predictable and acceptable in large databases. Apart from the development of effective and efficient data mining algorithms for various data mining tasks, many important issues, such as data mining methodologies, data mining processes, and integrated data mining environments remain to be studied further [Chen, 1999].

As Hand et al. [Hand et al., 2001] explain, data mining is the science of extracting useful information from large data sets, lying at the intersection of statistics, machine learning, data management and databases, pattern recognition, artificial intelligence, and other areas. It can also be considered as the task of discovering interesting patterns from large amounts of data where the data can be stored in database, data warehouse, or other information repositories [Thuraisingham, 1999].

Statistics has been used for data analysis for many years. Over the past years, much related statistics work has concentrated on hypothesis testing or confirmation [Long et al., 1991] [Glaymour et al., 1996]. Some estimation techniques [Chowdhury, 1991] in statistics have also been used in sampling data and dealing with missing values. In addition, exploratory data analysis is one of the important techniques for data classification and clustering. A complete survey of statistical perspectives on knowledge discovery and data mining can be found in [Elder and Pregibon, 1996].

Machine learning is most closely linked with data mining, forming the foundation for much of the work in the area of knowledge discovery in databases. Decision tree learning and rule induction [Quinlan, 1986] [Quinlan, 1993] is the major machine learning technique used for data classification in today's data mining. Neural networks have also been used as important tools for building predictive models in data mining [Lu et al., 1996]. Inductive logic programming (ILP) [Muggleton and De Raedt, 1994] is a new machine learning technique based on a first-order language. Recent advances in inductive logic programming [Džeroski, 1996], such as multiple predicate learning, inductive data engineering and causal discovery, have drawn great

attention in KDD. Differing from data mining, machine learning is based on the assumption that training data sets can be accommodated in the memory.

Databases relate to data mining obviously. Ad hoc querying and report generation supported by relational database management systems (DBMSs) are the most common tools for traditional data analysis. The newly-developed on-line analytical processing (OLAP) techniques [Harinarayan et al., 1996] provide users with a new way to manipulate and analyse data in data warehouses, using multi-dimensional methods. With OLAP, it is possible to collect statistics and aggregations based on various groupings of the data. Supporting operations from the database perspective in data mining systems is an emerging research area in the database community [Imieliński and Mannila, 1996] [Imieliński et al., 1996] [Chaudhuri, 1998].

Many methods and techniques have been presented by researchers from different research communities [Thuraisingham, 1999] [Han and Kamber, 2001] [Hand et al., 2001] [Chen, 1999] [Agrawal et al., 1993] [Cios et al., 1998], each technique typically suiting some problems better than others. As Chen [Chen, 1999] states, there is no universal data mining method and choosing a particular algorithm for a specific application is something of an art. The most important methods of data mining are classification, clustering, summarisation, regression, sequences and association rules.

As Han and Kamber state in [Han and Kamber, 2001], classification is the process of finding a set of models or functions that describe and distinguish data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown. Clustering analyses data objects without consulting a known class label. The objects are clustered or grouped based on the principle of "maximizing the intraclass similarity and minimizing the interclass similarity" [Han and Kamber, 2001]. Since summarisation is an approach towards characterising data via a small number of features or attributes, which means one or more sub-sets of the data are described according to a more general characteristic of this sub-set, it is often applied to an interactive exploratory data analysis and automated report generation. Regression involves predicting the value of a given continuous valued variable, based on the values of other variables, assuming a linear or non-linear model of dependency.

Sequences analysis identifies each item in a set of items as associated with its own timeline of events, and then discovers the dependencies among different events. The discovery of association rules shows the attribute-value conditions that occur frequently together in a given set of data. In other words, association rules are used to identify possible relationships among different record sets or tables within a database.

1.1.2 Temporal Data Mining

One of the main unresolved problems that arise during the data mining process is treating data that contains temporal information. In this case, a complete understanding of the entire phenomenon requires that the data should be viewed as a sequence of events [Antunes and Oliveira, 1998]. The attributes related to the temporal information present in this type of database need to be treated differently from other kinds of attributes. However, as Antunes and Oliveira [Antunes and Oliveira, 1998] point out, most of the data mining techniques tend to treat temporal data as an unordered collection of events, ignoring its temporal information. Therefore, the ability to record temporal data in database has created a new mine for knowledge derivation, further expanding data mining to temporal data.

Depending on the nature of the event sequence, the approach to solve the problem may be quite different. A sequence composed of a series of nominal symbols from a particular alphabet is usually called a temporal sequence; and a sequence of continuous, real-valued elements, is known as a time series [Antunes and Oliveira, 1998]. Both time series and temporal sequences appear naturally in a variety of different domains, which are described in [Antunes and Oliveira, 1998] as follows:

> *In engineering, time series and temporal sequences usually arise with either sensor-based monitoring or log-based systems monitoring. In scientific areas, they appear in spatial missions or genetics domain. In finance, applications on the analysis of product sales or inventory consumptions are of great importance to business planning [Antunes and Oliveira, 1998].*

Another very common application in finance is the prediction of the evolution of financial data. In healthcare, temporal sequences are a reality for decades; with data originated by complex data acquisition systems, or even with simple ones like measuring the patient temperature or treatments effectiveness [Antunes and Oliveira, 1998].

Temporal data mining is defined by [Chen, 1999] as "a set of approaches to deal with the problem of knowledge discovery from temporal data or database". The ultimate goal of temporal data mining is to discover hidden relations between sequences and sub-sequences of events. The knowledge obtained through temporal data mining is very important today, and forms one of the main focus fields of data mining [Chen, 1999] [Chen and Petrounias, 1998a] [Antunes and Oliveira, 1998] [Li et al., 1999] [Ale and Rossi, 2000] [Jensen, 1995] [Saraee and Theodoulidis, 1995].

1.2 Thesis Objectives

Many forms of patterns have been identified by data mining researchers using various methods. Association rules, motivated by the application of the market basket analysis, may be applied to various business domains like catalogue design and store layout to help people gain relevant business knowledge in order to improve the quality of decision-making. As an extension to association rule problems, temporal issues of association rules have been recently addressed in [Chen et al., 1998], [Chen and Petrounias, 1999], [Chen and Petrounias, 2000], [Chen and Petrounias, 2000a], [Özden et al., 1998], [Ramaswamy et al., 1998].

The mining of temporal association rules has a two-dimensional solution space, that is, the space consisting of patterns and temporal features. According to different restrictions, the problems of mining temporal association rules can be classified into three groups which are: finding the temporal features of a given association rule, finding association rules with a given temporal feature, and finding all possible temporal association rules of a certain kind [Chen et al., 1998]. Most research has been focused on the first two problems because it was thought that from a practical point of view, it was too expensive to directly search for all possible hidden temporal

association rules from large databases without any given information, and hence, the third problem group has not been studied so far. Consequently, this thesis focuses on that third problem, and tries to directly search for temporal association rules from temporal transaction datasets.

Previous research work such as that of [Chen, 1999] and [Chen and Petrounias, 1998a] developed several techniques for discovering the temporal features of a given association or discovering associations with given temporal clues. Additionally, these researchers believe that speculations can often be made by experienced experts, and need only to be validated before being used for decision-making. However, that is not always the case. A real world application also needs to discover unimaginable temporal association rules without any speculation. Experts make their speculations based on the knowledge they already know, and data mining is the technique that helps in discovering that knowledge, so, experts can not always make speculations before the knowledge has been discovered.

This thesis will consider the unsolved challenge of temporal association rule mining techniques, that being to directly discover temporal association rules, through two fields. The first study focus is on directly searching interval association rules from temporal transaction datasets, which is the main problem of directly mining temporal association rules. The second area of this thesis studies a tree projection algorithm, which will accelerate most itemset-based temporal association rule mining methods.

1.3 Thesis Organisation

This introductory chapter has provided the background to the research topic, through a brief discussion of data mining, and temporal data mining. It has then presented the objectives of the research.

Chapter Two - Literature Review: Data Mining

The purpose of Chapter Two is to present and further analyse the area of investigation for this research. First of all, the motivation for data mining is introduced in order to

establish its origin. Then, data mining is defined and briefly discussed before attention is turned to data mining approaches, the data mining process, and data mining techniques. Two data mining approaches are presented and six mining techniques are briefly considered.

Chapter Three - Literature Review: Association Rule Mining

As one of the most important techniques of data mining, and the research concern of this thesis, association rule mining forms the basis of Chapter Three. The definition of association rules is presented at the start of the chapter, and this is followed by the definition of frequent itemset, which is the core of association rule mining. Several algorithms, including AIS, SETM, Apriori, AprioriTid, AprioriHybrid, Partition and Sampling are then discussed in detail, and a comparison of these algorithms is given to present the best features of each. Finally, as most Apriori-based algorithms use the process of 'generating and counting candidate itemsets', an alternative way of mining frequent itemsets without candidate generation and mining association rules using logical operations is introduced at the end of the chapter.

Chapter Four - Literature Review: Temporal Data Mining

As the other concern of this thesis, temporal data mining forms the main content of Chapter Four and appears to support the research area of temporal association rule mining. The chapter details how the data mining process is extended to extract knowledge through temporal data, and clarifies issues related to the temporal data mining field. Firstly, an overview of temporal data mining is given to establish the background. Then, the motivation of temporal data mining, which is the potential knowledge in temporal databases, is discussed. Previous research studies about temporal data mining are then classified into four groups, these being: association rules, classification, clustering, and prediction. This is done in order for the next two sections to be able to clearly discuss temporal data mining, temporal features and representation of temporal features. Finally, the literature relating to temporal association rule mining techniques is discussed.

Chapter Five - Discovering Interval Association Rules

As one of the research areas of this thesis, an algorithm called IARMiner (Interval Association Rule Miner) that directly searches interval association rules from temporal transaction datasets, is discussed in Chapter Five. Firstly, since most of the background concerning interval association rule mining problems is discussed in previous chapters, some special issues about interval association rules are highlighted with the definitions relating to the interval association rule mining technique. Then, the algorithm IARMiner is introduced through the longest interval searching technique and association rule searching technique. The following section provides the actual description of the algorithm, before the section on implementation introduces the method and structure for realising IARMiner at some key points. Finally, the efficiency and effectiveness of IARMiner are evaluated by using both synthetic and real business datasets.

Chapter Six - A Tree Projection Method for Mining Temporal Association Rules

Chapter Six is the other research area of this thesis. It delivers a tree projection method called TI-tree to project temporal transaction datasets into a tree structure, which will accelerate the temporal association rule mining process. After introducing the idea of using a projected tree to speed up the mining process, Chapter Six discusses the tree structure and the special requirements of its use. Then, the method of mining temporal association rules from a projected tree produces the requirements for realising a tree, after which an improved tree structure is introduced to overcome some of the disadvantages. After the discussion on implementation, which presents the key program structures of this tree projecting method, the evaluation exercise uses both real business and synthetic datasets to evaluate the efficiency and effectiveness of the method.

Chapter Seven - Conclusion and Future Work

The purpose of Chapter Seven is to provide a summary of the research performed in this thesis, highlight the main points, and propose ideas for future research. Thus, a

comprehensive picture of the entire thesis is presented, outlining the main issues covered throughout. Additionally, based on the evaluations of Chapter Five and Chapter Six, potential improvements to the two algorithms are discussed with a view identifying further research possibilities.

Appendixes - two appendixes present the actual code of the programs which are realised to evaluate the algorithm IARMiner and the TI-tree. These two programs are coded using the Java programming language on the platform of Microsoft Visual J++ 6.0. The programs only realised the algorithms for evaluation, and not for a real application.

2 Data Mining

2.1 Introduction

Data mining is a relatively new technology that in recent years has been recognised as a promising new area for database research [Thuraisingham, 1999]. This chapter will describe data mining technology in several different aspects: data mining motivation, data mining and knowledge discovery in databases definition, data mining approaches, the data mining process, and data mining techniques. All these aspects will be summarised and some important research issues and challenges will be discussed. In this way, a complete and comprehensive understanding of the data mining area is provided to support the major concern of this thesis.

2.2 Motivation of Data Mining

Database technology has been characterised as employing advanced data models such as extended-relational, object-oriented, object-relational, and deductive models since the mid-1980s [Thuraisingham, 1999]. Issues related to the distribution, diversification, and sharing of data have been studied extensively. Han and Kamber [Han and Kamber, 2001] note that heterogeneous database systems and Internet-based global information systems such as the World Wide Web have emerged and play a vital role in the information industry. In addition, they point out that the amazing progress of computer hardware technology provides a great boost to the database and information industry, and makes a huge number of databases and information repositories available for transaction management, information retrieval, and data analysis.

Data can be stored in many different types of databases. One database architecture that has emerged is the data warehouse, which includes data cleaning, data integration, and On-Line Analytical Processing (OLAP) [Hand et al., 2001]. Furthermore, as Thuraisingham [Thuraisingham, 1999] mentions, a number of additional data analysis tools are required for in-depth analysis, such as data classification, clustering, and the characterisation of data changes over time.

However, the abundance of data, coupled with the need for powerful data analysis tools has been described as a 'data rich but information poor' situation [Hand et al., 2001]. Han and Kamber [Han and Kamber, 2001] observe that "data collected in large databases become 'data tombs', important decisions are often made based not on the information-rich data stored in databases but rather on a decision maker's intuition, because the decision maker does not have the tools to extract the valuable knowledge embedded in the vast amounts of data".

For this reason, the widening gap between data and information calls for the systematic development of data mining tools that will turn data tombs into 'golden nuggets' of knowledge [Han and Kamber, 2001]. As Hand et al. [Hand et al., 2001]

explain, data mining tools are able to perform data analysis and uncover important data patterns, contributing greatly to business strategies, knowledge bases, and scientific and medical research.

2.3 Data Mining

2.3.1 Knowledge Discovery in Databases

As Thuraisingham [Thuraisingham, 1999], Han and Kamber [Han and Kamber, 2001], and Hand et al. [Hand et al., 2001] state, data mining refers to extracting or mining knowledge from large amounts of data. However, many people treat data mining as a synonym for another popularly-used term, Knowledge Discovery in Databases (KDD). Alternatively, others view data mining as an essential step in the process of knowledge discovery in databases [Thuraisingham, 1999] [Han and Kamber, 2001].

Definitions for each are as follows [Fayyad et al., 1996]:

> *Data Mining is a step in the KDD process consisting of particular data mining algorithms that, under some acceptable computational efficiency limitations, produce a particular enumeration of patterns over databases [Fayyad et al., 1996].*

> *The KDD process uses data mining methods (algorithms) to extract what is deemed knowledge according to the specifications of measures and thresholds, using the database along with any required preprocessing, sub-sampling, and transformation of database [Fayyad et al., 1996].*

In this way, KDD refers to the overall process of discovering useful knowledge from data, while Data Mining refers to the application of algorithms for extracting patterns from data without the additional steps of the KDD process, such as incorporations of appropriate prior knowledge and proper interpretation of the results.

2.3.2 Data Mining

Thuraisingham [Thuraisingham, 1999] defines data mining as follows:

> *Data mining is the process of posing various queries and extracting useful information, patterns, and trends often previously unknown from large quantities of data possibly stored in database [Thuraisingham, 1999].*

Hand et al. [Hand et al., 2001] also describe data mining as the automatic exploration and analysis of huge amounts of data in order to extract meaningful patterns or rules that are hidden in a large database. Examples of such patterns or rules might be the association among the items that are purchased by customers in a supermarket database, an increasing trend in the price of a share in a stock market database, and the common characteristic of symptoms among a group of patients recorded in a medical database.

As Chen [Chen, 1999] argues, the potential applications of data mining can be grouped into three different domains: business, science and medicine. Chen [Chen, 1999] states that data mining is being used in a variety of business areas, such as marketing, finance, banking, insurance, etc. as many applications have shown the value of data mining in improving abilities to compete in business. In the scientific domain, a tremendous amount of raw data is collected from observations, experiments and simulations, but there is a widening gap between the abilities to collect that data and the resources to analyse it. Consequently, data mining is playing an increasingly more important role in the analysis of scientific data in various disciplines. Chen [Chen, 1999] also points out that a great number of medical databases, where a huge amount of knowledge may be potentially held, have been built. By using data mining technology, many research results have been reported on extracting different kinds of knowledge from various medical databases, for example, healthcare and patient-record databases.

Agrawal et al. [Agrawal et al., 1993] conclude that data mining is a new technology with great potential to help people uncover important information hidden in their databases or data warehouses.

2.4 Data Mining Approaches

As Berry and Linoff [Berry and Linoff, 1997], Thuraisingham [Thuraisingham, 1999], and Chen [Chen, 1999] explain, there are two approaches which can implement the data mining process on a database: 'top-down' and 'bottom-up' methodologies. Berry and Linoff [Berry and Linoff, 1997] state that these methodologies are neither the outcomes nor the techniques, but are rather, the steps one would take to perform the mining.

2.4.1 Top-down Approach

The top-down approach starts with a hypothesis proposed by domain experts. The hypothesis is validated or denied by testing it with the data in the database, and hence, the methodology is also known as hypothesis testing.

Agrawal et al. [Agrawal et al., 1993] state that hypothesis testing is about generating ideas, developing models, and then evaluating the model to determine whether the hypothesis is valid. The user begins with a hypothesis and uses data to refute or confirm it. Consequently, the rationale for the specific approach is that after the hypothesis is set, a series of techniques will be used to mine the data and test or prove the validity of the hypothesis. The most common techniques applied in this case concern statistical analysis, such as regression modelling, and multi-dimensional analysis.

2.4.2 Bottom-up Approach

The bottom-up approach is also known as knowledge discovery [Thuraisingham, 1999] [Han and Kamber, 2001] [Berry and Linoff, 1997]. It is different from the top-down approach as no initial hypothesis is set. As Thuraisingham [Thuraisingham,

1999] mentions, "this is much harder as the tool has to examine the data and then come up with patterns". In this way, the mining process is initialised in the database to discover knowledge that was previously unknown. Popular techniques and methods implemented in this approach include clustering, decision trees, summarisation, association rules, and neural models. In general, these techniques are applied to examine the data and identify meaningful patterns that are unknown in advance.

As Thuraisingham [Thuraisingham, 1999] illustrates, a combination of both top-down and bottom-up mining, named the 'hybrid approach', is possible. The mining process can be initialised with a bottom-up approach to identify unknown patterns, and thereafter a hypothesis can be set for these patterns. Finally a top-down approach can be used to test the hypothesis, and thus, the tool can switch between top-down and bottom-up mining.

2.5 Data Mining Process

Data mining forms a specific step within an overall process for discovering knowledge in databases. In several cases, it can also be identified as a process in its own right [Srikant and Agrawal, 1996] [Thuraisingham, 1999] [Chen, 1999]. A series of different attempts have been made to adequately describe all the different steps involved in the data mining process.

The data mining process involves acquiring all necessary knowledge relating to the client, and concentrating on a particular domain of data. Chen [Chen, 1999] provides a generalised view of the data mining process, showing the main activities being divided into three essential steps. Firstly, the data is prepared by determining a specific target source, applying the potential data transformation, and representing the cleaned data in a reflective and understandable way. Following this, the actual data mining occurs by choosing the corresponding techniques and algorithms to discover the patterns of hidden information, and then achieving the interesting outcomes. Finally, results explanation and interpretation is performed by analysing and evaluating the patterns obtained in the previous steps.

Fayyad et al. [Fayyad et al., 1996] present a more detailed data mining process, which is comprised of a series of nine fundamental sequential activities: clean or pre-process the target data, identify the potential reflective views for the data, define and select the desired data outcomes by implementing the appropriate algorithms, generate and represent the interesting patterns using an explicit method, and finally, assemble and organise the obtained knowledge.

Although several different approaches regarding the steps included in the data mining process exist, all follow a similar rationale and attempt to meet certain requirements in order for the process to be efficient and effective. They all require a specific, well-defined and clean set of data on which mining will be applied, determination of the desired outcomes, selection of the proper techniques and methods, and finally accurate results interpretation and evaluation.

Based on this summarisation, a descriptive model of the data mining process is provided in Figure 2.1.

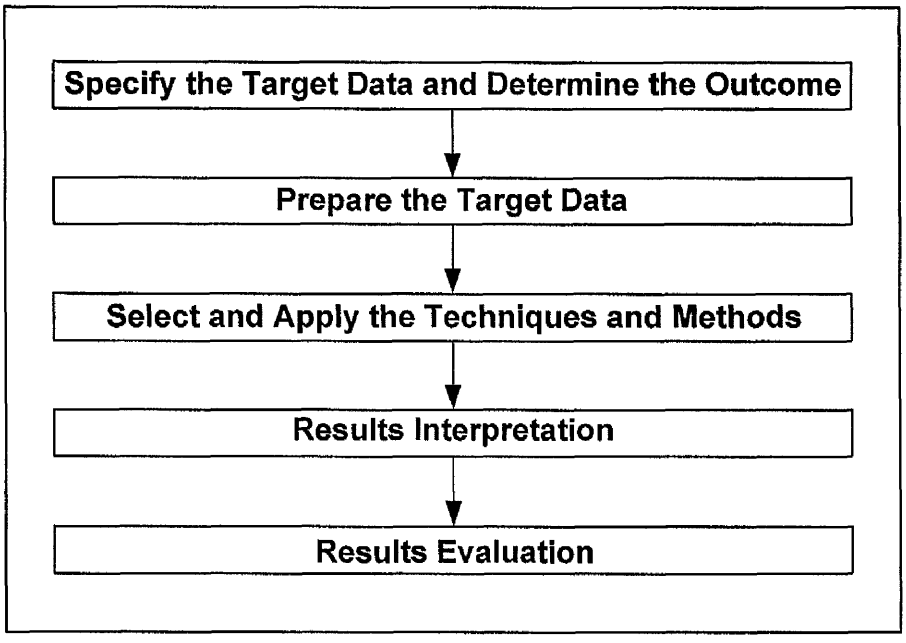


Figure 2.1 Data mining process model

Step 1: Specify the Target Data and Determine the Outcome

This step identifies the internal or external sources of information, and selects which sub-set of the data is needed for the data mining. As Hand et al. [Hand et al., 2001] point out, the objectives drive the entire data mining process. This step also defines the objectives of the mining process and specifies patterns.

Step 2: Prepare the Target Data

This step involves data sampling and quality testing. It performs potential joins, removes nulls and duplicates, and corrects invalid entries. Its aim is to ensure the quality of the selected data.

Step 3: Select and Apply the Techniques and Methods

This step selects the appropriate algorithms to accomplish the outcomes. A number of techniques and methods can be selected, for example, classification, association rules, clustering, summarisation, etc.

Step 4: Results Interpretation

This step maps knowledge acquired and perceives results. It interprets and evaluates the output according to the data mining operation. This step often requires a return to the data preparation stage.

Step 5: Results Evaluation

This step closes the whole loop. There are two challenges in this step, one is to present the new findings in a user-oriented way; the other is to formulate the ways in which the new information can be best exploited.

2.6 *Mining Techniques*

Many methods and techniques have been presented by researchers from different research communities [Thuraisingham, 1999] [Han and Kamber, 2001] [Hand et al., 2001] [Chen, 1999] [Agrawal et al., 1993] [Cios et al., 1998], each technique typically suiting some problems better than others. As Chen [Chen, 1999] states, there is no universal data mining method and choosing a particular algorithm for a specific application is something of an art. In the following sub-section, discussions are focused on just some of the data mining techniques for several typical tasks in the database community.

2.6.1 Classification

The purpose of classification is to develop training sets with pre-classified examples, and then build a model that fits the description of the classes. This model is then applied to the data not yet classified and results are obtained. As Han and Kamber [Han and Kamber, 2001] state, classification is the process of finding a set of models or functions that describe and distinguish data classes or concepts, for the purpose of being able to use the model to predict the class of objects whose class label is unknown. The derived model is based on the analysis of a set of training data, for which the class label is known.

Data classification is widely used in credit approval, target marketing, medical diagnosis, treatment effectiveness analysis, performance prediction, etc. An example in medical diagnosis is the classification of a set of diseases and the provision of symptoms which describe each class or sub-class.

2.6.2 Clustering

In this case, clustering analyses data objects without consulting a known class label. The objects are clustered or grouped according to the principle of "maximizing the intraclass similarity and minimizing the interclass similarity" [Han and Kamber,

2001]. In other words, the clusters are determined by examining the attributes of the relations in terms of continuity, or statistically, and also depending on the requirements of each case and the relationships among the data. As Thuraisingham [Thuraisingham, 1999] states, "clustering is a challenging field of research where its potential applications pose their own special requirement, the measures used in each case are determined depending on the requirements of each case and the relationships among the data".

The main difference between clustering and classification is that the clusters are created according to certain similarities characterising the group of records, whereas the classes are pre-defined.

2.6.3 Summarisation

As Thuraisingham [Thuraisingham, 1999] describes, summarisation provides a more abstract summary of the sub-set of data, and can be the mean or the standard deviation of a particular record set. Han and Kamber [Han and Kamber, 2001] also mention that summarisation is often applied to an interactive exploratory data analysis and automated report generation. In more detail, summarisation is an approach towards characterising data via a small number of features or attributes, which means one or more sub-sets of the data are described according to a more general characteristic of the sub-set. An example of summarisation is where a number of different salaries are described according to a mean approximation for different ages of employees.

2.6.4 Regression

Regression involves predicting a value of a given continuous valued variable, based on the values of other variables, assuming a linear or non-linear model of dependency. Cios et al. [Cios et al., 1998] point out that the extent, to which the regression model properly describes the data, depends on the character of data. In other words, regression is used in cases where the predicted output can take on many, or unlimited possible values. The underlying idea of regression is to construct a linear or non-linear function: $y = f(x, a)$ to explain the data.

2.6.5 Sequences

Sequence analysis identifies each item in a set of items as associated with its own timeline of events, and then discovers the dependencies among different events. In this way, the deviation or the activities of a process over time are revealed. In other words, sequence rules can be viewed as a special case of association rules, which mine the frequently-occurring patterns related to time or other sequences. This technique is useful for targeted marketing, customer retention, and weather prediction, as many business transactions are time-sequence data. As Han and Kamber [Han and Kamber, 2001] discuss, a sequence analysis could show the time when a specific item was purchased in relation to certain other relating goods. The same results could be generated for events or any type of record whose occurrence is temporarily related with other records.

2.6.6 Association Rules

The discovery of association rules shows the attribute-value conditions that occur frequently together in a given set of data. In other words, association rules are used to identify possible relationships among different record sets or tables within a database, expressed according to certain degrees of support and confidence. The support of a rule is the portion of transactions that the rule concerns. The confidence of a rule is the probability that a transaction containing X also contains Y. Assuming m is the number of transactions in D that contain X, n is the number of transactions in D that contain $X \cup Y$, and k is the number of transactions in D, the confidence c of a rule $X \Rightarrow Y$ in the transaction set D can be calculated by $c = m/n$, and the support s of a rule $X \Rightarrow Y$ in the transaction set D is evaluated by $s = n/k$.

An association rule between two items X and Y means that the presence of X in a record also implies the presence of Y in the same record. The discovery of such association helps retailers to develop marketing strategies by gaining insights into which items are frequently purchased together by customers. Association rules analysis is widely used for market basket or transaction data analysis, and also helps in many business decision-making processes.

2.7 Summary

The emphasis in this chapter has been on basic data mining concepts and techniques for uncovering interesting data patterns hidden in large data sets. Firstly, this chapter demonstrated how data mining is part of the natural evolution of database technology, why data mining is important, and how it is defined. Thereafter, the different approaches and the issues involved in the data mining process were reported and explained. Furthermore, because data mining involves various techniques and representation methods, a discussion of these primary techniques was presented.

3 Association Rule Mining

3.1 Introduction

This chapter focuses on association rules analysis, and will explore the concept of association rules. The discovery of interesting association relationships among huge amounts of business transaction records can help in many business decision-making processes. A typical example is market basket analysis, which helps the retailer to develop marketing strategies by gaining insight into which items are frequently purchased together by customers.

The discovery of frequent itemsets will be presented next. This step generates the candidate itemsets and counts their support. Because the overall performance is determined in this step, most algorithms focus on reducing the numbers of generated candidates and the numbers of scans at this point. Following this, a number of algorithms will be introduced for discovering significant association rules between items in a large database of transactions, among them AIS, SETM, Apriori, AprioriTid, AprioriHybrid [Agrawal and Srikant, 1994], Partition [Savasere et al., 1995] and Sampling [Toivonen, 1996]. Additionally, an alternative way of mining association rules without generating candidates, called FP-growth [Han et al., 2000], and a special method of mining association rules using logical operations, named PRICES [Wang and Tjortjis, 2004] will be presented.

3.2 Association Rules

Mining association rules is one of the important data mining applications [Agrawal and Srikant, 1994] [Agrawal et al., 1993a] [Srikant and Agrawal, 1996]. Since the concept of association rules was first introduced by Agrawal for the 'market-basket' analysis purpose, the discovery of association rules has been extensively studied in several ways. There are numerous applications that fit into this framework, the supermarket being a canonical example. The items are products and the baskets are customer purchases at the checkout. By using association rules analysis, the supermarket manager can determine what products customers are likely to buy together, which can be very helpful for planning and marketing.

In general, as Agrawal and Srikant [Agrawal and Srikant, 1994] describe, association rule analysis focuses on a database that consists of a large collection of transactions, where each transaction is a collection of data items. Discovering association rules refers to the identification of relationships between the appearances of various items within the sub-sets (baskets). Agrawal and Srikant [Agrawal and Srikant, 1994] also show that for a given set of transactions, the problem of mining association rules is to generate all association rules that have support and confidence greater than the user-specified minimum support and minimum confidence respectively. 'Support' represents the portion of transactions in a set of transactions, and 'confidence' denotes the probability that a transaction that contains X will also contain Y [Agrawal and Srikant, 1994] [Agrawal et al., 1993a] [Srikant and Agrawal, 1996] [Bayardo and Agrawal, 1999] [Sarawagi et al., 1998]. The confidence of a rule reveals how often it can be expected to apply, while its support indicates the trustworthiness of the entire rule. For a rule to be relevant, enough support and sufficient confidence are necessary.

Agrawal and Srikant [Agrawal and Srikant, 1994] define association rules as follows:

> *"Let* $I = \{i_1, i_2, \cdots, i_m\}$ *be a set of literals, called items. Let* D *be a set of transactions, where each transaction* T *is a set of items such that* $T \subseteq I$*. We say that a transaction* T *contains* X*, a set of some items in* I*, if* $X \subseteq T$*. An*

association rule is an implication of the form $X \Rightarrow Y$, *where* $X \subset I$, $Y \subset I$ *and* $X \cap Y = \varnothing$. *The rule* $X \Rightarrow Y$ *holds in the transaction set* D *with confidence* c *if* $c\%$ *of transactions in* D *that contain* X *also contain* Y. *The rule* $X \Rightarrow Y$ *has support* s *in the transaction set* D *if* $s\%$ *of transactions in* D *contain* $X \cup Y$ *" [Agrawal and Srikant, 1994].*

For example, if there are two sets of items X and Y, and an association rule between them is an expression of the form $X \Rightarrow Y$, denoting the presence of X in one transaction will also imply the presence of Y in the same transaction [Agrawal and Srikant, 1994] [Agrawal et al., 1993a] [Srikant and Agrawal, 1996]. An example of an association rule could be given in a student bookshop database, showing that 25% (support) of students who purchase science books also buy fiction books, with a probability of 85% (confidence). In this case, X = 'science books', Y = 'fiction books', there are 25% of transactions that contain the items science books and fiction books, and 85% of transactions that have the item science books also have the item fiction books in them.

3.3 Discovering Frequent Itemsets

The association rules will be generated from the frequent itemsets. As Megiddo and Srikant [Megiddo and Srikant, 1999] describe, the general idea of generating association rules is that if AB and A are both frequent itemsets, one can determine that the rule $A \Rightarrow B$ holds, if the value of $confidence(AB)$ is greater than the minimum confidence.

Most association rule algorithms follow a basic scheme. However, different algorithms separate rule construction from finding frequent itemsets. This section will focus on the basic role of counting frequent itemsets used in most of the algorithms. In the next section, algorithms for finding frequent itemsets will be discussed.

3.3.1 Frequent Itemset Discovery

As Agrawal and Srikant [Agrawal and Srikant, 1994] state, the algorithms for discovering frequent itemsets make multiple passes over the data. In the first pass, the algorithms count the support of individual items and determine which of them are frequent. The basic method which every algorithm follows is to create a set of itemsets, called candidates. In each subsequent pass, the algorithms start with a seed set of itemsets found to be frequent in the previous pass. The algorithms use this seed set for generating new potentially frequent itemsets. These are candidate itemsets. The algorithm counts the actual support for the candidate itemsets during the pass over the data. At the end of the pass, the algorithm determines which of the candidate itemsets are actually frequent, and they become the seed for the next pass. This process continues until no new frequent itemsets are found.

As counting the occurrences of a candidate set affects the amount of the processing memory and time, reducing the number of candidates generated by the algorithm and the number of passes over the database becomes the goal of the performance optimisation.

3.3.2 Association Rule Construction

Possible association rules can be generated and their confidence can be determined, as long as all frequent sets and support values are available. The minimum confidence level determines whether the rule is accepted or discarded. For example, if AB is a frequent itemset, the $confidence(A \Rightarrow B) = \frac{support(AB)}{support(A)}$ determines if the rule $A \Rightarrow B$ holds. If $confidence(A \Rightarrow B) \geq minimum\ confidence$, the rule holds. If $confidence(A \Rightarrow B) < minimum\ confidence$, the rule fails, and no sub-sets of its antecedent need to be further considered.

3.4 Association Rule Algorithms

3.4.1 AIS

The concept of association rules was first introduced by Agrawal et al. [Agrawal et al., 1993a], who then discussed the AIS algorithm in [Agrawal and Srikant, 1994]. Candidate itemsets are generated and counted on the fly as the database is scanned. After reading a transaction, it is determined which of the itemsets that were found to be frequent in the previous pass, are contained in this transaction. Extending these frequent itemsets with other items in the transaction generates new candidate itemsets. These frequent itemsets are called frontiers, and the candidates are called extensions.

A frequent itemset l is extended with only those items that are frequent and occur later in the lexicographic ordering of items more frequent than any of the items in l. The extensions are generated when a transaction that supports a frontier set is read. The candidates created from a transaction are added to the set of candidate itemsets maintained for the pass, or the counts of the corresponding entries are increased if they were created by an earlier transaction. After a complete pass through all transactions, the candidates with more frequent than the minimum support, become the new frontier sets, the next extension phase begins and continues until no frontier sets have been found. This implies that none of the previous candidates are frequent.

However, the AIS algorithm does not lead to efficient performance, mainly because it creates a large number of candidates, and uses a sophisticated pruning technique to decide if an extension should be included in the candidate set. Further details of the AIS algorithm can be found in [Agrawal and Srikant, 1994].

3.4.2 SETM

The SETM algorithm [Agrawal and Srikant, 1994] was motivated by the desire to use SQL to compute frequent itemsets. As with the AIS algorithm, SETM generates candidates on the fly based on transactions read from the database. However, SETM uses the standard SQL join operation for candidate generation. It uses its own data

representation to store every itemset supported by a transaction along with the transaction's ID (TID).

As Agrawal and Srikant [Agrawal and Srikant, 1994] state, SETM separates candidate generation from counting. It saves a copy of the candidate itemset together with the TID of the generating transaction in a sequential structure. At the end of the pass, the support count of candidate itemsets is determined by sorting and aggregating this sequential structure. The SQL code of SETM is shown in Figure 3.1.

insert *into* C_{k+1}
select $a.TID, a.item_1, a.item_2, \ldots, a.item_k, b.item_k$
from $L_k a, L_k b$
where $a.TID = b.TID, a.item_1 = b.item_1, \ldots, a.item_{k-1} = b.item_{k-1}, a.item_k < b.item_k$

Figure 3.1 SQL code of SETM [Agrawal and Srikant, 1994]

The advantage of SETM is that it remembers the TIDs of the generating transactions with the candidate itemsets. SETM can easily find the frequent itemsets contained in the transaction read, and therefore, the sub-set operation is avoided. Agrawal and Srikant [Agrawal and Srikant, 1994] point out that SETM only needs to visit every member of L_k once in the TID order and the candidate generation can be performed by using the relational merge-join operation.

The disadvantage of this algorithm is due to the size of candidate sets. For each candidate itemset, the candidate set has as many entries as the number of transactions in which the candidate itemset is present. Moreover, these huge relations have to be sorted twice on the TID before they can be used for generating candidates in the next pass. In this way, the inefficiencies of SETM have outweighed its advantage.

3.4.3 Apriori

This, and the following sections discuss two algorithms for finding association rules: Apriori and AprioriTid, that are fundamentally different to AIS and SETM in terms of which candidate itemsets are counted in a pass, and in the way that those candidates are generated.

Apriori was proposed by Agrawal and Srikant in [Agrawal and Srikant, 1994]. It separates candidate generation from support counting, which causes two major effects. Firstly, it reduces the number of generated candidates. Secondly, it increases efficiency in the support counting step. For this reason, most of the more recent association rule algorithms are variations of Apriori.

The pseudo code of Apriori (Figure 3.2) is taken from Agrawal and Srikant [Agrawal and Srikant, 1994], who state that the first pass of the algorithm simply counts item occurrences to determine the frequent $1-itemsets$. A subsequent pass consists of two phases.

First, the frequent itemsets L_{k-1} found in the $(k-1)th$ pass are used to generate the candidate itemsets C_k, by using the apriori-gen function.

Next, the database is scanned and the support of candidates in C_k is counted. For fast counting, it is necessary to efficiently determine the candidates in C_k that are contained in a given transaction t.

```
1)  L_1 = {frequent 1-itemsets};
2)  for(k = 2; L_{k-1} ≠ ∅; k++) do begin
3)      C_k = apriori-gen(L_{k-1}); //New candidates
4)      forall transactions t ∈ D do begin
5)          C_t = subset(C_k, t); //Candidates contained in t
6)          forall candidates c ∈ C_t do
7)              c.count++
8)      end
9)      L_k = {c ∈ C_k | c.count ≥ minsup}
10) end
11) Answer = ⋃_k L_k;
```

Figure 3.2 Pseudo code of Apriori [Agrawal and Srikant, 1994]

Apriori candidate generation: apriori-gen

The apriori-gen function consists of two steps [Agrawal and Srikant, 1994]: the *join* step and the *prune* step. It takes as argument L_{k-1}, the set of all frequent

$(k-1)-itemsets$. In other words, it returns a superset of the set of all frequent $k-itemsets$. As Agrawal and Srikant [Agrawal and Srikant, 1994] state, apriori-gen is successful in reducing the number of candidates that has been used in most proposed algorithms.

In the *join* step, apriori joins L_{k-1} with L_{k-1} to generate potential candidates. As Agrawal and Srikant [Agrawal and Srikant, 1994] state, this step is equivalent to extending L_{k-1} with each item in the database and then deleting those itemsets for which the $(k-1)-itemset$ obtained by deleting the $(k-1)th$ item is not in L_{k-1}. As shown in Figure 3.3, the condition $p.item_{k-1} < q.item_{k-1}$ simply ensures that no duplicates are generated. Thus, after the join step, $L_k \subseteq C_k$. For example, let L_3 be $\{\{1\,2\,3\},\{1\,2\,4\},\{1\,3\,4\},\{1\,3\,5\},\{2\,3\,4\}\}$. After the join step, C_4 will be $\{\{1\,2\,3\,4\},\{1\,3\,4\,5\}\}$.

insert into C_k
select $p.item_1, p.item_2, \cdots, p.item_{k-1}, q.item_{k-1}$
from $L_{k-1}\ p, L_{k-1}\ q$
where $p.item_1 = q.item_1, \cdots, p.item_{k-2} = q.item_{k-2}, p.item_{k-1} < q.item_{k-1}$;

Figure 3.3 Join step of apriori-gen [Agrawal and Srikant, 1994]

In the prune step, apriori deletes all itemsets $c \in C_k$, such that some $(k-1)-sub\text{-}set$ of c is not in L_{k-1}, which is presented in Figure 3.4. In other words, all $(k-1)-sub\text{-}sets$ of each candidate are tested for membership in L_{k-1}. If one sub-set does not exist in L_{k-1}, the candidate is deleted. Using the same example as above, this step will delete the itemset $\{1\,3\,4\,5\}$ because the itemset $\{1\,4\,5\}$ is not in L_3, and then leave with only $\{1\,2\,3\,4\}$ in C_4.

forall *itemsets* $c \in C_k$ do
  forall $(k-1)-subsets\ s\ of\ c$ do
    if $(s \notin L_{k-1})$ then
      delete c from C_k;

Figure 3.4 Prune step of apriori-gen [Agrawal and Srikant, 1994]

The sub-set function increases the efficiency in the support count step. A set of candidate $k-itemsets$ C_k and a transaction t are considered as input, and the set of $k-itemsets$ that are contained in the transaction C_t are given as output.

Candidate itemsets C_k are stored in a hash-tree. Agrawal and Srikant [Agrawal and Srikant, 1994] discussed the sub-set function in detail. A node of the hash-tree either contains a list of itemsets (a leaf node) or a hash table (an interior node). In an interior node, each bucket of the hash table points to another node. The root of the hash-tree is defined to be at depth 1. An interior node at depth d points to nodes at depth $d+1$. Itemsets are stored in the leaves. When an itemset c is added, it starts from the root and goes down the tree until it reaches a leaf. At an interior node at depth d, a branch is followed by applying a hash function to the d^{th} item of the itemset. All nodes are initially created as leaf nodes. When the number of itemsets in a leaf node exceeds a specified threshold, the leaf node is converted to an interior node.

3.4.4 AprioriTid

The interesting feature of AprioriTid is that the database D is not used for counting support after the first pass. Rather the set C_k in the previous pass is used for this purpose. Apriori has to read the entire database in each pass, although many items and transactions are no longer needed in later passes. Counting the sets of the impossible candidates causes expensive effort in performance. For this reason, AprioriTid employs an encoding of the candidate itemsets for this purpose. In the later passes, the size of this encoding can become much smaller than the database, thus saving much reading effort.

Agrawal and Srikant [Agrawal and Srikant, 1994] present AprioriTid that also uses the apriori-gen function to determine the candidate itemsets before the pass begins. Each member of the set C_k' is of the form $\langle TID, \{X_k\} \rangle$, where each X_k is a potentially frequent $k-itemset$ present in the transaction with identifier TID.

For $k=1$, C_1' corresponds to the database D, although conceptually each item i is replaced by the itemset $\{i\}$.

For $k>1$, C_k' is generated by the algorithm. The member of C_k' corresponding to transaction t is $\langle t.TID, \{c \in C_k | c \text{ contained in } t\}\rangle$.

If a transaction does not contain any candidate $k-itemset$, then C_k will not have an entry for this transaction. Thus, the number of entries in C_k' may be smaller than the number of transactions in the database, especially for large values of k.

For large values of k, each entry may be smaller than the corresponding transaction because very few candidates may be contained in the transaction.

For small values of k, each entry may be larger than the corresponding transaction because an entry in C_k includes all candidates $k-itemsets$ contained in the transaction.

3.4.5 AprioriHybrid

As Agrawal and Srikant [Agrawal and Srikant, 1994] point out, Apriori and AprioriTid use the same candidate generation procedure and, therefore, count the same itemsets. In the earlier pass, Apriori does better than AprioriTid. However, in the later passes, the number of candidate itemsets reduces, but Apriori still examines every transaction in the database. On the other hand, rather than scanning the database, AprioriTid scans C_k' for obtaining support counts, and the size of C_k' has become smaller than the size of the database. In this way, AprioriTid beats Apriori in later passes.

Based on such observation, Agrawal and Srikant design a hybrid algorithm, called AprioriHybrid, that uses Apriori in the initial passes and switches to AprioriTid when it expects that the set C_k' at the end of the pass will fit in memory. The size of the C_k'

should be estimated if C'_k would fit in memory in the next pass. If C'_k in this pass is small enough to fit in memory, and there are fewer frequent candidates in the current pass than in the previous pass, it can switch to AprioriTid.

In general, the advantage of AprioriHybrid over Apriori depends on how the size of the C'_k set declines in the later passes, as switching from Apriori to AprioriTid does involve a cost. Agrawal and Srikant [Agrawal and Srikant, 1994] point out that AprioriHybrid may incur the cost of switching without realising the benefits in some particular situations. For example:

If C'_k remains large until nearly the end and then has an abrupt drop, it will not gain much by using AprioriHybrid, since it can use AprioriTid only for a short period of time after the switch.

If there is a gradual decline in the size of C'_k, AprioriTid can be used for a while after the switch. A significant improvement can be obtained in the execution time.

3.4.6 Apriori-like Algorithms Comparison

In both AIS and SETM, candidate itemsets are generated on the fly during the pass as data is being read. Specifically, after reading a transaction, it is determined which of the itemsets found to be frequent in the previous pass are present in the transaction. As Agrawal and Srikant [Agrawal and Srikant, 1994] point out, the major disadvantage is that this results in unnecessarily generating and counting too many candidate itemsets that turn out to be small.

Apriori and AprioriTid differ fundamentally from AIS and SETM in terms of which candidate itemsets are counted in a pass, and in the way that those candidates are generated. Agrawal and Srikant [Agrawal and Srikant, 1994] describe how Apriori and AprioriTid generate the candidate itemsets to be counted in a pass by using only the itemsets found to be frequent in the previous pass – without considering the transactions in the database. This results in the generation of a much smaller number

of candidate itemsets. In addition, AprioriTid does not use database for counting the support of candidate itemsets after the first pass. Rather, it employs an encoding of the candidate itemsets used in the previous pass for this purpose. As the size of this encoding can become much smaller than the database, it saves much reading effort.

Furthermore, the best features of Apriori and AprioriTid can be combined into AprioriHybrid. AprioriHybrid scales linearly with the number of transactions, and has excellent scale-up properties with respect to the transaction size and the number of items in the database. It opens up the feasibility of mining association rules over very large databases.

3.4.7 Partition

Unlike the Apriori-like level-wise algorithms, Savasere et al. introduced an association rule algorithm called Partition in [Savasere et al., 1995], which reads the database at most two times to generate all significant association rules. As Figure 3.5 shows, algorithm Partition divides the database D into small non-overlapping partitions $p_1, p_2, \ldots, p_n$ and considers them one at a time. The partition sizes are chosen such that each partition can be accommodated in the main memory so that the partitions are read only once in each phase, and there will be no additional disk I/O for each partition after being loaded into the main memory.

Partition executes in two phases. In the first phase, the local frequent itemsets L_i in each partition $p_i (1 \le i \le n)$, can be found by using a level-wise algorithm such as Apriori. At the end of the first phase, these frequent itemsets $L_i (1 \le i \le n)$ are merged to generate the global candidate set C^G. It uses the property that a frequent itemset in the whole database must be locally frequent in at least one partition of the database [Savasere et al., 1995]. In the second phase, all candidates in C^G are counted through the whole database, the actual supports for these candidates are generated, and the frequent itemsets are identified.

$$
\begin{aligned}
&P = partition\_database(D)\\
&n = Number\ of\ partitions\\
&for\ i = 1\ to\ n\ begin\ //\ Phase\ 1\\
&\quad read\_in\_partition(p_i \in P)\\
&\quad L^i = Apriori(p_i)\\
&end\\
&C^G = \bigcup_{i=1,2,\ldots,n} L^i\ //\ Merge\ Phase\\
&for\ i = 1\ to\ n\ begin\ //\ Phase\ 2\\
&\quad read\_in\_partition(p_i \in P)\\
&\quad for\ all\ candidates\ c \in C^G\\
&\qquad gen\_count(c, p_i)\\
&end\\
&L^G = \left\{c \in C^G \middle| c.count \geq min\_sup\right\}
\end{aligned}
$$

Figure 3.5 Algorithm Partition [Savasereet al., 1995]

Partition favours a homogeneous data distribution [Savasere et al., 1995]. That is, if the count of an itemset is evenly distributed in each partition, most of the itemsets to be counted in the second scan will be large. However, for a skewed data distribution, most of the itemsets in the second scan may turn out to be small, thus wasting a lot of CPU time counting false itemsets. As Savasere et al. pointed out [in Savaere et al., 1995], the effect of data skew can be reduced by randomising the data allocated to each partition. This is done by choosing the data to be read in a partition randomly from the database.

3.4.8 Sampling

Sampling [Tovivonen, 1996], is another association rule algorithm trying to reduce the I/O overhead for very large databases. It reduces the number of database scans to one in the best case and two in the worst. The idea of sampling is to pick a random sample, which can fit in the main memory, to find by using this sample, all frequent itemsets that probably hold in the whole database, and then to verify the results with the rest of the database.

Sampling makes use of the concept of negative border. For example [Toivonen, 1996] [Mannila and Toivonen, 1996], let the items in the dataset $R = \{A, B, \ldots, F\}$ and assume the collection F_S of frequent itemset is $\{A\}$, $\{B\}$, $\{C\}$, $\{F\}$, $\{A,B\}$, $\{A,C\}$, $\{A,F\}$, $\{C,F\}$, $\{A,C,F\}$. The negative border of this collection contains, e.g., the set $\{B,C\}$, which is not in the collection F_S, but all its subsets are. The whole negative border of F_S is $Bd^-(F_S) = \{\{B,C\},\{B,F\},\{D\},\{E\}\}$. The intuition behind the concept is that given a closed frequent itemset, the negative border contains the "closest" itemsets that could be frequent, too [Toivonen, 1996]. The negative border function is a generalisation of the apriori-gen function in Apriori [Toivonen, 1996]. When all itemsets in F_S are of the same size, $Bd^-(F_S) = apriori\_gen(Fs)$. The difference lies in that the negative border can be applied to a set of itemsets of different sizes, while the function $apriori\_gen(\ )$ only applies to a single size.

But as Mannila and Toivonen pointed out in [Mannila and Toivonen, 1996], the importance is the fact that the negative border needs to be evaluated, in order to be sure that no frequent itemsets are missed. To illustrate this, suppose $F_S = \{\{A\},\{B\},\{C\},\{A,B\}\}$. The candidate itemsets for the first scan are $Bd^-(F_S) \cup F_S = \{\{A,C\},\{B,C\}\} \cup \{\{A\},\{B\},\{C\},\{A,B\}\} = \{\{A\},\{B\},\{C\},\{A,B\},\{A,C\},\{B,C\}\}$. If the frequent itemsets are $\{\{A\},\{B\},\{C\},\{A,B\},\{A,C\},\{B,C\}\}$, i.e., there are two misses $\{A,C\}$ and $\{B,C\}$ in $Bd^-(Fs)$, the itemset $\{A,B,C\}$, which might be frequent, but not counted in the first scan of Sampling. Hence the Sampling algorithm needs one more scan to count the new candidate itemsets like $\{A,B,C\}$. The new candidate itemsets are generated by applying the negative border function recursively to the misses. The algorithm is shown in Figure 3.6.

$Ds = a\ random\ sample\ drawn\ from\ D$ // *draw a sample*

$Fs = Apriori(Ds, min\_sup)$ // *find local frequent itemsets in the sample*

$C = Bd^{-}(Fs) \cup Fs$

$count(C, D)$ // *first scan counts the candidates generated from Fs*

$Fm = \{x | x \in Bd^{-}(Fs), x.count \geq min\_sup \times |D|\}$ // *Fm are the misses*

if $Fm \neq \varnothing$ *then* // *Cm are the new candidates generated from the misses*

$Cm = \{x | x \in C, x.count \geq min\_sup \times |D|\}$

repeat

$Cm = Cm \cup Bd^{-}(Cm)$

until Cm does not grow

$Cm = Cm - C$ // *itemsets in C have already been counted in the first scan*

$count(Cm, D)$ // *second scan counts additional candidates*

return $F = \{x | x \in C \cup Cm, x.count \geq min\_sup \times |D|\}$

Figure 3.6 Algorithm Sampling [Toivonen, 1996]

The entire approach is divided into two phases. During the first phase a sample of database is obtained and all frequent itemsets in the sample are found by using a level-wise algorithm such as Apriori. Let the set of frequent itemsets in the sample be F_S. The candidates are generated by applying the negative border function, $Bd^{-}(\ )$, to F_S. Thus the candidates are $Bd^{-}(Fs) \cup Fs$. After the candidates are generated, the whole database is scanned once only to determine the counts of the candidates. If all frequent itemsets are in F_S, i.e., no itemsets in $Bd^{-}(Fs)$ turn out to be frequent, then all frequent itemsets are found and the algorithm terminates. Otherwise, i.e. there are misses in $Bd^{-}(Fs)$, some new candidate itemsets must be counted to ensure that all frequent itemsets are found in the second phase, and thus a second scan is needed.

Sampling can find association rules very efficiently in only one database pass. Experiments show that this method works very well in practice, making the approach attractive especially for very large databases [Toivonen, 1996]. But the potential problem is that the algorithm Sampling needs the datasets to be input as binary schemas. A binary schema guarantees that all items in a dataset appear in the chosen sample. Thus, no item can be missed in both F_S and $Bd^{-}(F_S)$.

3.5 Mining Frequent Itemsets without Candidate Generation

In many cases the Apriori candidate generate-and-test method reduces the amount of candidate itemsets significantly and leads to a good performance. However, it may suffer from two costs [Han and Kamber, 2001].

- Firstly, it may need to generate a huge number of candidate itemsets. For example, if there are 10^4 frequent $1-itemsets$, the Apriori algorithm will need to generate more than 10^7 candidate $2-itemsets$ and accumulate and test their occurrence frequencies. Moreover, to discover a frequent itemset of size 100, such as $\{a_1,\ldots,a_{100}\}$, it must generate more than $2^{100} \approx 10^{30}$ candidates in total.

- Secondly, it may need to repeatedly scan the database and check a large amount of candidates by itemset matching. This is especially the case for mining long itemsets.

3.5.1 Frequent Pattern Growth

An interesting method called frequent-pattern growth (FP-growth) [Han et al., 2000], can mine the complete set of frequent itemsets (or called frequent patterns) without candidate generation, which adopts a divide-and-conquer strategy as follows:

> *Compress the database representing frequent items into a frequent-pattern tree, or FP-tree, but retain the itemset association information, and then divide such a compressed database into a set of conditional databases, each associated with one frequent item, and mine each such database separately [Han and Kamber, 2001].*

Table 3.1 shows a transaction database D to be mined using the frequent-pattern growth approach.

TID	List of item_IDs
T100	I1, I2, I5
T200	I2, I4
T300	I2, I3
T400	I1, I2, I4
T500	I1, I3
T600	I2, I3
T700	I1, I3
T800	I1, I2, I3, I5
T900	I1, I2, I3

Table 3.1 Transaction dataset for FP-growth [Han and Kamber, 2001]

The first scan of the database is the same as Apriori, which derives the set of frequent $1-itemsets$ and their support counts. Suppose the minimum support is 2. The set of frequent items is sorted in the order of descending support count. Thus, the frequent $1-itemsets$ is $L=[I2:7,\quad I1:6,\quad I3:6,\quad I4:2,\quad I5:2]$.

An FP-tree is then constructed as follows [Han et al., 2000] [Han and Kamber, 2001]. First, create the root of the tree, labelled '*null*'. Scan database D a second time. The items in each transaction are processed in L order and a branch is created for each transaction. For example (Figure 3.7), the scan of the first transaction, "$T100:I1,I2,I5$", which contains three items $(I2,I1,I5)$ in L order, leads to the construction of the first branch of the tree with three nodes: $\langle(I2:1),(I1:1),(I5:1)\rangle$, where $I2$ is linked as a child of the root, $I1$ is linked to $I2$, and $I5$ is linked to $I2$. The second transaction, $T200$, contains the items $I2$ and $I4$ in L order, which would result in a branch where $I2$ is linked to the root and $I4$ is linked to $I2$. However, this branch would share a common prefix, $\langle I2\rangle$, with the existing path for $T100$. Therefore, the count of the $I2$ node is increased by 1, and a new node, $(I4:1)$, which is linked as a child of $(I2:2)$ is created.

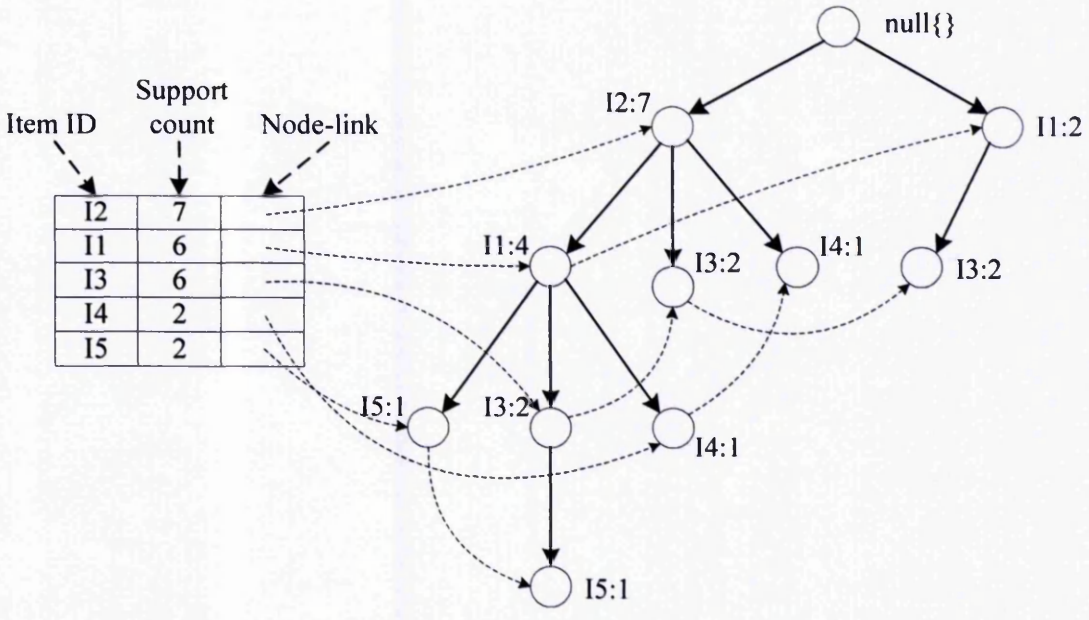


Figure 3.7 FP-tree [Han and Kamber, 2001]

In general, when considering the branch to be added for a transaction, the count of each node along a common prefix is incremented by 1, and nodes for the items following the prefix are created and linked accordingly.

To facilitate tree traversal, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links (Figure 3.7). Therefore, the problem of mining frequent itemsets in databases is transformed to that of mining the FP-tree.

The mining of the FP-tree proceeds as follows [Han et al., 2000] [Han and Kamber, 2001]. Start from each frequent $1-itemset$ (as an initial suffix itemset), construct its conditional itemset base (a 'sub-database' which consists of the set of prefix paths in the FP-tree co-occurring with the suffix itemset), then construct its (conditional) FP-tree, and perform mining recursively on such tree. The itemset growth is achieved by the concatenation of the suffix itemset with the frequent itemsets generated from a conditional FP-tree.

Mining of the FP-tree is summarised as shown in Table 3.2. Firstly, consider $I5$ which is the last item in L, rather than the first. $I5$ occurs in two branches of the FP-tree (Figure 3.7). The paths formed by these branches are $\langle(I2 \quad I1 \quad I5:1)\rangle$

and $\langle(I2 \quad I1 \quad I3 \quad I5:1)\rangle$. Therefore, considering $I5$ as a suffix, its corresponding two prefix paths are $\langle(I2 \quad I1:1)\rangle$ and $\langle(I2 \quad I1 \quad I3:1)\rangle$, which form its conditional itemset base. Its conditional FP-tree contains only a single path, $\langle I2:2, \quad I1:2\rangle$; $I3$ is not included because its support count of 1 is less than the minimum support threshold. The single path generates all the combinations of frequent itemsets: $I2 \quad I5:2$, $I1 \quad I5:2$, $I2 \quad I1 \quad I5:2$.

item	*conditional itemset base*	*conditional FP-tree*	*frequent itemsets generated*
I5	{(I2 I1: 1), (I2 I1 I3: 1)}	<I2: 2, I1: 2>	I2 I5: 2, I1 I5: 2, I2 I1 I5: 2
I4	{(I2 I1: 1), (I2: 1)}	<I2: 2>	I2 I4: 2
I3	{(I2 I1: 2), (I2: 2), (I1: 2)}	<I2: 4, I1: 2>, <I1: 2>	I2 I3: 4, I1 I3: 2, I2 I1 I3: 2
I1	{(I2: 4)}	<I2: 4>	I2 I1: 4

Table 3.2 Mining of FP-tree [Han and Kamber, 2001]

For $I4$, its two prefix paths form the conditional itemset base, $\{(I2 \quad I1:1), \quad (I2:1)\}$, which generates a single-node conditional FP-tree $\langle I2:2\rangle$ and derives one frequent itemset, $I2 \quad I4:2$. Although $I5$ follows $I4$ in the first branch, there is no need to include $I5$ in the analysis here since any frequent itemset involving $I5$ has been analysed in the examination of $I5$. This is the reason that processing started at the end of L, rather than at the front.

Similar to the above analysis, $I3$'s conditional itemset base is $\{(I2 \quad I1:2), \quad (I2:2), \quad (I1:2)\}$. Its conditional FP-tree has two branches, $\langle I2:4, \quad I1:2\rangle$ and $\langle I1:2\rangle$, as shown in Figure 3.8, which generates the set of itemsets: $\{I2 \quad I3:4, \quad I1 \quad I3:2, \quad I2 \quad I1 \quad I3:2\}$. Finally, $I1$'s conditional itemset base is $\{(I2:4)\}$, whose FP-tree contains only one node $\langle I2:4\rangle$, which generates one frequent itemset, $I2 \quad I1:4$.

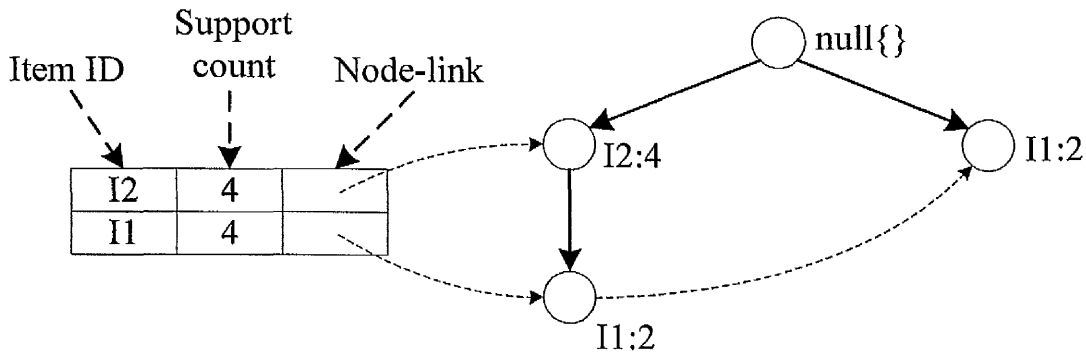


Figure 3.8 Conditional FP-tree [Han and Kamber, 2001]

The mining process of this FP-growth method is summarised in Figure 3.9 and Figure 3.10.

FP - tree construction process.
1) Scan the transaction database D once. Collect the set of frequent items F and their supports.
2) Sort F in support descending order as L, the list of frequent items.
3) Create the root of an FP - tree, and label it as " null".
4) For each transaction Trans in D do
4.1) Select and sort the frequent items in Trans according to the order of L.
4.2) Let the sorted frequent item list in Trans be $[p|P]$, where p is the first element and P is the remaining list.
4.3) Call insert_tree$([p|P], T)$*, which is performed as follows.*
4.3.1) If T has a child N such that N.item − name = p.item − name, then increment N' s count by 1;
4.3.2) else create a new node N, and let its count be 1, its parent link be linked to T, and its node - link to the nodes with the same item - name via the node - link structure.
4.3.3) If p is nonempty, call insert_tree(P, N) *recursively.*

Figure 3.9 Algorithm of FP-tree construction [Han and Kamber, 2001]

FP - tree mining process
procedure FP_growth$(Tree, \alpha)$
1) if Tree contains a single path P then
2) for each combination (denoted as β) of the nodes in the path P
3) generate itemset $\beta \cup \alpha$ with support = minimum support of nodes in β;
4) else for each a_i in the header of Tree {
5) generate itemset $\beta = a_i \cup \alpha$ with support = a_i.support;
6) construct β' s conditional itemset base and then β' s conditional FP - tree $Tree_\beta$;
7) if $Tree_\beta \neq \varnothing$ then
8) call FP_growth$(Tree_\beta, \beta)$*; }*

Figure 3.10 Algorithm of FP-tree mining [Han and Kamber, 2001]

The FP-growth method transforms the problem of finding long frequent itemsets, looking for shorter ones recursively and then concatenating the suffix. It uses the least frequent items as a suffix, offering good selectivity. The method substantially reduces the search costs.

When the database is large, it is sometimes unrealistic to construct a main memory-based FP-tree. An interesting alternative is to first partition the database into a set of projected database, and then construct an FP-tree and mine it in each projected database [Han and Kamber, 2001]. Such a process can be recursively applied to any projected database if its FP-tree still cannot fit in main memory.

A study on the performance of the FP-growth method shows that it is efficient and scalable for mining both long and short frequent itemsets, and is about an order of magnitude faster than the Apriori algorithm. It is also faster than a Tree-Projection algorithm which projects a database into a tree of projected databases recursively [Han and Kamber, 2001].

3.6 PRICES

The number of times an algorithm scans the entire database is a significant factor in terms of speed as it determines the number of time-consuming I/O operations involved. PRICES [Wang and Tjortjis, 2004] reduces frequent itemset generation time, known to be the most time-consuming step, by scanning the database only once and using logical operations in the process.

PRICES uses the same two steps as in other algorithms; it is however faster as it scans the database only once, to store transactions information in the memory by a succinct form called Prices Table [Wang and Tjortjis, 2004]. This Prices table can be pruned by creating a pseudo transaction table called Pruned Prices Table, which contains all $1-size$ frequent itemsets after eliminating all those $1-size$ non-frequent ones. PRICES generates $k-size$ frequent itemsets from the Pruned Prices Table and $(k-1)-size$ frequent itemsets instead of scanning the dataset. It uses logical

operations, such as *AND*, *OR*, *XOR* and *left − shift* in the process of generating frequent itemsets and association rules, thus accelerating the mining process.

PRICES gives every item in the transactions a unique value. So every transaction can be represented by a price, which is the sum of the item values it consists of. Therefore, every price represents a unique itemset pattern. For example [Wang and Tjortjis, 2004], if there are 5 items, A, B, C, D and E in a database, let the value of item A be $2^4 = 10000$ in binary mode, the value of item B $2^3 = 01000$ and so on (Table 3.3). The price of transaction $T_1 = \{A, C, D\}$ will be 10110 in binary mode (Table 3.4). In this way, an itemset can also be represented as a price such as $Price(\{A, C\}) = P_{AC} = 10100$.

item	*value*
A	10000
B	01000
C	00100
D	00010
E	00001

Table 3.3 Values of items [Wang and Tjortjis, 2004]

TID	*items*	*price*
T_1	ACD	$P_{T_1} = 10110$
T_2	BCE	$P_{T_2} = 01101$
T_3	$ABCE$	$P_{T_3} = 11101$
T_4	BE	$P_{T_4} = 01001$

Table 3.4 Prices of transactions [Wang and Tjortjis, 2004]

Under this assumption, logical operation *AND* can be applied to the price of one transaction and the price of one itemset to determine whether this transaction contains the itemset, by comparing the result with the itemset price. For example [Wang and Tjortjis, 2004], the transaction $T_1 = \{A, C, D\}$ ($P_{T_1} = 10110$) contains itemset $\{A, C\}$ ($P_{AC} = 10100$) because P_{AC} AND $P_{T_1} = P_{AC}$. Therefore, the task of counting itemsets

changes to identifying all the itemsets prices from $\{00...01\}$ to $\{11...11\}$ occurring above a threshold in the prices of transactions.

3.6.1 Frequent Itemset Generation

The PRICES algorithm generates frequent itemsets in three steps. First, the Pruned Prices Table is created, then all frequent $2-itemsets$ are created and finally, all frequent itemsets are generated.

Pruned Prices Table

Since it is known that any itemset which contains a small itemset will also be small [Agrawal and Srikant, 1994], a Prices Table can be pruned by eliminating the column of small items [Wang and Tjortjis, 2004]. This is done in two steps: first generate the Large Bit Mark (LBM), which is the price of the itemset containing all frequent $1-itemsets$; then create the Pruned Prices Table. To generate the LBM one can set the price of the first $1-size$ candidate to 1 and apply the $left-shift$ operation to generate the second candidate price and so on. After calculating each $1-size$ candidate's support, if a candidate is frequent, the corresponding position in LBM is set to 1, otherwise to 0 (Table 3.5, assume $min\_sup = 50\%$). In addition, the frequent $1-size$ itemsets, along with the support and size, are stored in F (Table 3.6).

	A	*B*	*C*	*D*	*E*
support	2	3	3	1	3
compare with min_sup	≥	≥	≥	<	≥
LBM	1	1	1	0	1

Table 3.5 Large Bit Mark [Wang and Tjortjis, 2004]

frequent itemset	*support*	*size*	*pruned price*
A	2	1	$PP_A = 1000$
B	3	1	$PP_B = 0100$
C	3	1	$PP_C = 0010$
E	3	1	$PP_E = 0001$

Table 3.6 Frequent itemset table [Wang and Tjortjis, 2004]

Given the LBM and Prices Table, the Pruned Prices Table can be generated by eliminating the columns which have 0 in the corresponding position of LBM. Since a 0 in the LBM indicates that the corresponding item is non-frequent, and thus any itemset containing it is also non-frequent. Therefore, removing these items shrinks the Prices Table without affecting the generation of frequent itemsets [Wang and Tjortjis, 2004]. Table 3.7 shows how PP_{T_1} (the pruned price of transaction T_1) is generated from LBM and P_{T_1}; and Table 3.8 shows all pruned prices of 4 transactions in the example given by Table 3.4.

	A	B	C	D	E	
P_{T_1}	1	0	1	1	0	10110
LBM	1	1	1	0	1	11101
PP_{T_1}	1	0	1	–	0	1010

Table 3.7 Generating pruned price [Wang and Tjortjis, 2004]

TID	*price*	*pruned price*
T_1	$P_{T_1} = 10110$	$PP_{T_1} = 1010$
T_2	$P_{T_2} = 01101$	$PP_{T_2} = 0111$
T_3	$P_{T_3} = 11101$	$PP_{T_3} = 1111$
T_4	$P_{T_4} = 01001$	$PP_{T_4} = 0101$

Table 3.8 Pruned prices of transactions [Wang and Tjortjis, 2004]

Generate Frequent 2-itemsets

As every single item in the Pruned Prices Table is frequent, every $2-size$ itemset composed of a different single item from the Pruned Prices Table is a candidate. The

algorithm PRICES calculates the support of every candidate and if it is frequent, records it into F (Table 3.9), along with its support and size. To compose two different item prices into one price, PRICES uses the *OR* operation. For example [Wang and Tjortjis, 2004], the price of itemset $\{A,E\}$ ($P_{AE} = 10001$) can be derived by applying *OR* to itemset $\{A\}$ ($P_A = 10000$) and $\{E\}$ ($P_E = 00001$), such as $P_{AE} = P_A \ OR \ P_E = 10001$.

frequent itemset	*support*	*size*	*pruned price*
A	2	1	$PP_A = 1000$
B	3	1	$PP_B = 0100$
C	3	1	$PP_C = 0010$
E	3	1	$PP_E = 0001$
AC	2	2	$PP_{AC} = 1010$
BC	2	2	$PP_{BC} = 0110$
BE	3	2	$PP_{BE} = 0101$
CE	2	2	$PP_{CE} = 0011$

Table 3.9 Frequent itemset table [Wang and Tjortjis, 2004]

Generate k-size Itemsets

$k-size$ frequent itemsets can then be generated from $(k-1)-size$ frequent itemsets and the Pruned Prices Table. PRICES uses the *XOR* operation as a difference indicator from which one can find how many different bits (items) there are between two $(k-1)-size$ frequent itemsets. To generate a candidate C_k, two $(k-1)-size$ frequent itemsets must have exactly two different bits. Like Apriori, the candidates generation and count process repeats until all frequent itemsets are found. Table 3.10 shows the result of F generated from the example presented by Table 3.4.

frequent itemset	*support*	*size*	*pruned price*
A	2	1	$PP_A = 1000$
B	3	1	$PP_B = 0100$
C	3	1	$PP_C = 0010$
E	3	1	$PP_E = 0001$
AC	2	2	$PP_{AC} = 1010$
BC	2	2	$PP_{BC} = 0110$
BE	3	2	$PP_{BE} = 0101$
CE	2	2	$PP_{CE} = 0011$
BCE	2	3	$PP_{BCE} = 0111$

Table 3.10 Frequent itemset table [Wang and Tjortjis, 2004]

In order to get the frequent itemsets from F, PRICES restores the prices in F from pruned prices and maps those into itemsets. According to the definition of LBM, a 0 is inserted into pruned prices at corresponding positions to restore prices. Once the prices are restored, PRICES maps these into itemsets based on the previous definition of the relationship between price and itemset. Table 3.11 shows this phase.

pruned price	*price*	*frequent itemset*
1000	10$\underline{0}$0	A
0100	01$\underline{0}$0	B
0010	00$\underline{1}$0	C
0001	000$\underline{1}$	E
1010	10$\underline{1}$0	AC
0110	01$\underline{1}$0	BC
0101	01$\underline{0}$1	BE
0011	00$\underline{1}$1	CE
0111	01$\underline{1}$1	BCE

Table 3.11 Restore pruned prices [Wang and Tjortjis, 2004]

PRICES is an algorithm which mines association rules in two steps by the use of logical operations. Its major advantage is that it only scans the database once and any consecutive processing takes places in memory using logical operations. This advantage means that PRICES outperformes Apriori in terms of speed [Wang and Tjortjis, 2004]. As Wang and Tjortjis pointed out in [Wang and Tjortjis, 2004], PRICES triggers further research to address the memory requirements and

performance deterioration due to I/O overhead when data do not fit in memory due to the possibly very large size of datasets.

3.7 Summary

This chapter focuses on association rules analysis. The concept of association rules has been presented. The discovery of frequent itemsets was then presented. This step generates the candidate itemsets and counts their support. Because the overall performance is determined in this step, most algorithms focus on reducing the numbers of generated candidates and the numbers of scans during this step. Following this, a number of algorithms were introduced for discovering significant association rules between items in a large database of transactions, including AIS, SETM, Apriori, AprioriTid, AprioriHybrid, Partition and Sampling. Comparison of these algorithms was provided. Finally, an alternative way of mining association rules without generating candidates, called FP-growth and a method of mining association rules using logical operations, named PRICES were presented.

4 Temporal Data Mining

4.1 Introduction

When the data mining process involves treating data which contains temporal information, it forms a new challenging area of temporal data mining, because the special characteristics of temporal information require it to be treated differently from other kinds of information. This gives the common data mining problem another dimension that makes the difficulty more complicated. The temporal related problem arises in many areas such as engineering, science, finance, healthcare etc. This chapter focuses on many aspects of the temporal data mining problem. Firstly, the background of temporal data mining will be discussed in detail and the definition of temporal data mining will be considered. Then, to achieve the objective of understanding and formalising the problem, the potential knowledge in temporal databases will be introduced and the patterns in such databases will be classified. After that, temporal data mining operations will be discussed in the categories of association rules, classification, clustering and prediction. Next, for the following research work, the temporal features and the representation of these will be introduced. As the main concern of this thesis, the temporal association rule mining problem will be defined and discussed in detail.

4.2 Background to Temporal Data Mining

As Antunes and Oliveira [Antunes and Oliveira, 1998] state, with the rapid increase of stored data, the interest in the discovery of hidden information within the data has exploded in the last decade. The discovery of the hidden information has mainly been focused on data classification, data clustering and association finding, with the result that one of the main unresolved problems arising during the data mining process is treating data that contains temporal information. In this case, a complete understanding of the entire phenomenon requires that the data should be viewed as a sequence of events [Antunes and Oliveira, 1998].

The attributes related to the temporal information need to be treated differently from other kinds of attributes. However, as Antunes and Oliveira [Antunes and Oliveira, 1998] point out, most of the data mining techniques tend to treat temporal data as an unordered collection of events, ignoring its temporal information. Therefore, the ability to record temporal data in a database has created a new mine for knowledge discovery, further expanding data mining to temporal data.

Antunes and Oliveira [Antunes and Oliveira, 1998] further classify the temporal problem into two categories as follows:

> *Depending on the nature of the event sequence, the approach to solve the problem may be quite different. A sequence composed by a series of nominal symbols from a particular alphabet is usually called a temporal sequence and a sequence of continuous, real-valued elements, is known as a time series. Both time series and temporal sequences appear naturally in a variety of different domains.*

Additionally, they enumerate several areas in which temporal related problems arise, as for instance in engineering, time series and temporal sequences usually arise with either sensor-based monitoring or log-based systems monitoring. In scientific areas, they appear in spatial missions or the genetics domain. In finance, applications on the

analysis of product sales or inventory consumptions are of great importance to business planning.

Antunes and Oliveira [Antunes and Oliveira, 1998] also state that, as another very common application in finance, temporal issues are involved in the prediction of the evolution of financial data. In healthcare, temporal sequences have been a reality for decades, with data originated by complex data acquisition systems, or even with simple ones like measuring the patient temperature or treatments effectiveness.

Temporal data mining is defined by Chen [Chen, 1999] as "a set of approaches to deal with the problem of knowledge discovery from temporal data or database". The ultimate goal of temporal data mining is to discover hidden relations between sequences and sub-sequences of events. The knowledge obtained through temporal data mining is very important and, it forms one of the main fields of data mining [Chen, 1999] [Chen and Petrounias, 1998a] [Antunes and Oliveira, 1998] [Li et al., 1999] [Ale and Rossi, 2000] [Jensen, 1995] [Saraee and Theodoulidis, 1995].

4.3 Potential Knowledge in Temporal Databases

Lee et al. [Lee et al., 1998] state that temporal databases provide a complete history of all changes to a database and include the times when changes occurred. This allows users to query the current state of the database, as well as past states, and even planned future.

4.3.1 Data Objects in Temporal Databases

The data objects stored in temporal databases are the information sources of temporal data mining. A data object (e.g., an attribute value or a tuple) may be associated with a time instant or a time interval, depending on the type (an event or a state) of object that it expresses. From the view of conceptual modelling, data objects in temporal databases can be classified into the following types, according to their temporal characteristics [Zaniolo et al., 1997] [Etzion et al., 1998]:

- ***Time-Independent Objects***

  In information systems, there are some objects that do not change their values once they are created. These objects are usually not stamped with any time values.

- ***Time-Varying State Objects***

  In many applications, some objects may change values with arbitrary frequency. These objects are modelled as states in temporal databases. A state is an instance of temporal data that is satisfied or exists over a period of time or at a specific time instant. Each recorded state of a time-varying object is time-stamped with an interval to record the time during which the state is satisfied or a time instant to record the moment at which the state exists.

- ***Time-Series State Objects***

  These objects change their values and the change is tightly associated with a particular pattern of time. In application systems, they are usually modelled as a sequence of states. Each state is time-stamped with a time instant to record the moment at which the state exists.

- ***Time-Related Event Objects***

  An event object is an instance of temporal data that happens during an instant in time. In many systems, each recorded event is associated with a time-stamp to record the time instant during which the event occurs.

In a temporal database, data concerns information relating to three different chronological features involved with each record in a database: valid time, transaction time and user-defined time [Jensen, 1995] [Saraee and Theodoulidis, 1995] [Chen, 1999] [Chen and Petrounias, 2000].

- Valid time concerns modelling a time-varying reality, and refers to the actual time an event occurred, such as the specific day a product was purchased. It possibly spans the past, present, and future. However, Jensen [Jensen, 1995] points out that, the valid time of a fact may not necessarily be recorded in the

database, for any number of reasons. For example, the valid time may not be known, or recording it may not be relevant for the applications supported by the database. If a database models different possible worlds, the database facts may have several valid times, one for each such world.

- Transaction time concerns the moment at which the storage of information in the database occurs. Unlike valid time, transaction time may be associated with any database entity, not only with facts.

- User-defined time involves time intervals specified by the user according to the user's needs or interpretation of certain chronological facts. Jensen [Jensen, 1995], Saraee and Theodoulidis [Saraee and Theodoulidis, 1995] present the detailed description of valid time, transaction time and user-defined time in temporal database.

4.3.2 Identifying Patterns in Temporal Databases

Four main temporal pattern categories have been classified, [Han et al., 1992] [Agrawal and Srikant, 1995] [Chen, 1999] [Chen and Petrounias, 2000a] and [Han and Kamber, 2001], these being: inducted patterns, trends patterns, similar patterns and sequential patterns.

- ***Inducted Patterns***

  Inducted patterns arise through various inducted temporal rules that can be applied in a database. Chen [Chen, 1999] describes an inducted rule as a description or classification for a sub-set of data in the database. Such classification occurs based on certain common temporal features or relationships, which may exist among the elements of the record set. Characteristic rules, classification and clustering rules are some popular types of inducted rules. A characteristic rule, as described in [Han et al., 1992] [Chen and Petrounias, 2000a], is an assertion that characterises a concept satisfied by all or a majority number of the examples in the class undergoing learning, called the target class. For example, the symptoms of a specific disease can be

summarised by a characteristic rule. On the other hand, classification or clustering rules are created according to the common properties of the attributes of the objects, belonging to a pre-defined class or cluster [Chen, 1999].

- ***Trend Patterns***

Chen and Petrounias [Chen and Petrounias, 2000a] describe trends as particularly important knowledge forms in time series data, characterising the upward or downward fluctuations in a series of data over a period of time. They state that a trend pattern is the increasing or decreasing oscillation that can describe time series data during a specific period. In fact, trend patterns discovered through temporal data mining can yield significant temporal knowledge. In real world applications, trends are often used to predict the expected data values, which may form the references of deviation patterns for time-series data. For example, a trend could be that during 1990 the TOSHIBA fax-machine sales increased by 20% .

- ***Similar Patterns***

Agrawal et al. [Agrawal et al., 1993], and Chen [Chen, 1999] define Similar Patterns as time sequences that are similar or tend to be similar to a reference sequence in time series databases. Thus, similar 'movements' in data across the time interval are identified for examining similar patterns. For example, similar patterns can help a retailer who wants to optimise purchasing and store keeping, to find groups of products that have similar forecasted seasonal sales for the next year, and enable the retailers to use this information for combining purchases and inventory replenishment. As a result, identifying similar patterns with respect to time is significantly important. The process can be utilised in several areas to enhance future predictions, for example, in marketing, finance or medical areas.

- ***Sequential Patterns***

Saraee and Theodoulidis [Saraee and Theodoulidis, 1995] reveal another major source for database mining, this being the ordered data, such as temporal data related to stock, and point of sales data. For example, given a database of

customer transactions over a period of time, each transaction is a list of items in a visit and all transactions of a particular customer are temporally ordered. In this case, all the sequential buying patterns supported by a specified minimum fraction of customers could be looked for. Saraee and Theodoulidis [Saraee and Theodoulidis, 1995] present another example of a rule about stock market data, which could be "when BT stock goes up on 2 consecutive days and Mercury stock does not fall during this period, Orange stock goes up the next day 75% of the time". Another example would be a car insurer who wants to study lapsing and retention among his/her customers. By applying the sequential patterns technique, the insurer can understand what events lead to lapses.

In addition, sequence rules are handled in a similar way to the association rules. The temporal nature of relationships between antecedents and consequents can be handled by the TDBMS (Temporal Database Management System) [Saraee and Theodoulidis, 1995]. For example, a direct mailer who wants to maximise cross-selling opportunities can apply the Association and Sequential Patterns technique to historical order data, to establish what articles sell together and what articles are bought in a sequence over time [Saraee and Theodoulidis, 1995]. The mailer can use this information to decide on placements of articles in the catalogue and for deciding on a flyer to attach with a bill.

4.4 Temporal Data Mining Operations

Antunes and Oliveira [Antunes and Oliveira, 1998] presented a clear view of temporal data mining, saying that the ultimate goal of this process is to discover hidden relations between sequences and sub-sequences of events. As a result, the discovery of relations between sequences of events involves the application of models and representations to the actual mining problems.

4.4.1 Association Rules

One of the most common approaches to mining frequent patterns is the Apriori method [Agrawal and Srikant, 1994]. When a transactional database represented as a set of sequences of transactions performed by one entity is used, the manipulation of temporal sequences requires that some adaptations be made to the Apriori algorithm [Antunes and Oliveira, 1998]. There are several aspects to consider when applying temporal features to the association rule mining area [Antunes and Oliveira, 1998].

- ***Sequence***

  The most important modification of mining sequential association rules is on the notion of supports. Support is the fraction of entities, which had consumed the itemsets in any of their possible transactions [Agrawal and Srikant, 1995], i.e. an entity could only contribute one time to increment the support of each itemset, beside it could have consumed that itemset several times [Antunes and Oliveira, 1998].

  After identifying the frequent itemsets, the itemsets with support greater than the minimum support allowed, they are translated to an integer, and each sequence is transformed in a new sequence, whose elements are the frequent itemsets of the previous-one. The next step is to find the frequent sequences. For achieve this, the algorithm acts iteratively as Apriori: first it generates the candidate sequences and then it chooses the frequent sequences from the candidate ones, until there are no candidates [Antunes and Oliveira, 1998].

  Unlike the Apriori-based approaches, the method of searching frequent itemsets without candidate generation (FP-growth), [Han et al., 2000], follows a different method to deal with sequential data and is presented in [Han et al., 2000a].

- ***Relevant Association Rules***

  The discovery of relevant association rules is one of the most important methods used to perform data mining on transactional databases. Although Apriori is an

effective algorithm to discover association rules, adapting this method to deal with temporal information leads to some different approaches [Antunes and Oliveira, 1998].

A possible approach consists of extending the notion of a typical rule $X \Rightarrow Y$ to be a rule with new meaning: $X \Rightarrow^{T} Y$ (which states: if X occurs then Y will occur within time T) [Das et al., 1998]. Stating a rule in this new form, allows for controlling the impact of the occurrence of an event to the other event occurrence, within a specific time interval [Antunes and Oliveira, 1998].

- ***Periodical Association Rules***

  Another method consists of considering cyclic rules [Özden et al., 1998]. A cyclic rule is one that occurs at regular time intervals, i.e. transactions that support specific rules occur periodically, for example at every first Monday of a month. In order to discover these rules, it is necessary to search for them in a restricted portion of time, since they may occur repeatedly at specific time instants but on a little portion of the global time considered. A method to discover such rules is applying an algorithm similar to the Apriori, and after having the set of traditional rules, detects the cycles behind the rules [Antunes and Oliveira, 1998].

4.4.2 Classification

A classification method to deal with temporal sequences is based on the merge operator. This operator receives two sequences and returns "a sequence whose shape is a compromise between the two original sequences" [Keogh and Pazzani, 1998]. The basic idea is iteratively merging a typical example of a class with each positive example, building a more general model for the class.

Classification is relatively straightforward if generative models are employed to model the temporal data [Antunes and Oliveira, 1998]. Deterministic and probabilistic models can be applied in a straightforward way to perform classification [Lang et al.,

1998], since they give a clear answer to the question of whether a sequence matches a given model [Antunes and Oliveira, 1998].

4.4.3 Clustering

The fundamental problem in clustering temporal sequences databases consists of the discovery of a number of clusters, able to represent the different sequences. Antunes and Oliveira [Antunes and Oliveira, 1998] point out that there are two central problems in clustering: choosing the number of clusters and initialising their parameters. They also point out another important problem, which appears when dealing with temporal sequences, that is the existence of a meaningful similarity measure between sequences. A number of approaches have been considered in different ways e.g., [Smyth, 1999] [Smyth, 1997] [Ketterlin, 1997] [Fisher, 1987].

4.4.4 Prediction

Prediction is one of the most important problems in data mining. However, in many cases, prediction problems may be formulated as classification, association rule finding or clustering problems, but the prediction problems have some specific characteristics that differentiate them from other problems [Antunes and Oliveira, 1998]. Much literature exists on computer-assisted prediction of time series, in a variety of domains [Weigend and Gershenfeld, 1994], [Mannila et al., 1999], [Lu et al., 1998] [Giles et al., 2001].

As Antunes and Oliveira [Antunes and Oliveira, 1998] state, in the specific domain of prediction, care must be taken with the domain where prediction is to be applied. Well-known and generally accepted results on the inherent unpredictability of many financial time series imply that significant gains in prediction accuracy are not possible, no matter how sophisticated the techniques used.

4.5 Representation of Temporal Features

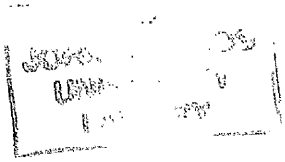
As a fundamental tool for temporal data mining research, the formal representation of temporal features is confidently expected to depict temporal aspects of potential knowledge in temporal data.

4.5.1 Time Representation

The physical time space is considered as a hierarchy of totally ordered sets of time intervals. The smallest, indivisible time unit is called a chronon, and the set of chronons is at the lowest level in the hierarchy. Chen and Petrounias [Chen, 1999] [Chen and Petrounias, 1998a] describe a chronon as "a non-decomposable time interval of some fixed, minimal duration, in which an event takes place". Each level is a partition of the next lower level with conventional meaning, such as 1 minute corresponds to 60 consecutive seconds. The time measure at each of the levels is called the basic granularity. Granularity refers to the time measure used for one or more sets within the time domain [Chen and Petrounias, 1998]. For example, the granularity of years is months, whilst the granularity of minutes is seconds. Granularity can vary according to the representation and measurement needs, for instance, the granularity of year can be months, days, or seconds.

The representation problem is especially important when dealing with time series, since direct manipulation of continuous, high-dimensional data in an efficient way is extremely difficult. Antunes and Oliveira [Antunes and Oliveira, 1998] address the problem in several different ways: time-domain continuous representations, transformation-based representation, and discretisation-based methods. However, [Chen, 1999] highlights several reasons for choosing a discrete time representation.

- Firstly, measures of time are inherently imprecise [Anderson, 1982] [Clifford and Tansel, 1985]. Clocking instruments invariably report the occurrence of events in terms of intervals (even with very small granularity), not time 'points'.



Hence, events, even so-called 'instantaneous events', can best be measured as having occurred during a time period.

- Secondly, most natural language references to time are compatible with the discrete time model. For example, when we say that an event occurred at 4.30 pm , we usually do not mean that the event occurred at the 'point' in time associated with 4.30 pm but at some time in the time period (perhaps a minute) associated with 4.30 pm [Anderson, 1982] [Clifford and Rao, 1987].

- Thirdly, the concepts of time points and intervals allow us to naturally model events that are not instantaneous but have duration [Anderson, 1982].

- Finally, any implementation of a data model with a temporal dimension will, of necessity, have to have some discrete encoding of time [Snodgrass, 1987].

4.5.2 Time Series

Time series is a powerful abstraction mechanism to handle collections of data that possess observed values at regular periods or intervals. Lee et al. [Lee et al., 1998] define a time series as "a collection of observations made sequentially over time". It consists of a sequence of events, an event being "an ordered pair consisting of a temporal value and an associated list of data value" [Lee et al., 1998]. Collection and analysis of financial data and scientific data, are examples that can be modelled with time series.

4.5.3 Calendar

Each data item can be of an atomic data type or a structured data type, such as record, list array, etc., which provides flexibility in modelling various types of complex data items. The sequence of temporal values is based on a calendar, which determines the granularity and period of events associated with a time series. Lee et al. [Lee et al., 1998] describe a calendar model as a totally-ordered set or sequence of intervals with additional semantics which is represented by a tuple

$\langle granularity, pattern, period, start\ time, end\ time \rangle$. Since time series in different applications can have different types of temporal values for their events, there are possibly many different types of calendars that can be utilised [Lee et al., 1998] [Chen, 1999] [Chen and Petrounias, 1998a]. Some familiar calendar systems are the Gregorian, Islamic and Oriental–lunar calendars.

4.6 Temporal Features

Based on the above concepts, two fundamental temporal features employed in the temporal data mining process should be introduced: time intervals and periodicity.

4.6.1 Time Intervals

Time at each level is represented by a temporal element that is a finite union of time intervals: $\{I_1, I_2, \ldots\}$ where I_i is a time interval. Lee et al. [Lee et al., 1998] define a time interval as "an ordered pair of time units represented by $[tl, tu]$, which refer to any specific set of consecutive time units with a given start time, end time and granularity". For example, the interval between September 1st and September 30th forms a time interval of the calendar unit 'months' with granularity days, whilst the calendar unit 'days' also consists of sub-intervals with granularity hours etc. [Chen, 1999].

4.6.2 Periodicity

Lee et al. [Lee et al., 1998] describe time period as "the length of a time interval at which a pattern occurs repeatedly". It is expressed in terms of the number of time units of a particular granularity in that period, or by a particular granularity. Antunes and Oliveira [Antunes and Oliveira, 1998] indicate that periodicity involves a cyclic rule that occurs at regular time intervals, i.e. transactions that support specific rules occurring periodically, for example, on every first Monday of a month. Such events (first Monday) are called periodic events, whilst such intervals (months) are called periodic intervals. In order to discover these rules, it is necessary to search for them in

a restricted portion of time, since they may occur repeatedly at specific time instants but on a little portion of the global time considered.

Periodicity analysis can be applied to many important areas, for example, seasons, tides, planet trajectories, daily power consumptions, daily traffic patterns, and weekly TV programs, all present certain periodic patterns. Han and Kamber [Han and Kamber, 2001] observe that mining periodic patterns can be viewed as mining sequential patterns by taking durations as a set of partitioned sequences, such as every year, every slot after or before the occurrence of certain events, and so on.

4.7 Temporal Association Rule Mining

In the traditional problem of association rule discovery, if the given minimum support and confidence are high, many rules may not be found from the database. However, it is possible that some rules may exist with satisfied support and confidence during a short period of time or a series of short periods (regularly or irregularly) of time over the whole time domain. To get more accurate and useful information, it is necessary to introduce temporal issues of association rules, which have been addressed in [Chen et al., 1998], [Özden et al., 1998], [Ramaswamy et al., 1998] [Chen and Petrounias, 1999], [Chen and Petrounias, 2000] and [Chen and Petrounias, 2000a].

4.7.1 Temporal Association Rules

The major concern in [Chen et al., 1998] is the discovery of association rules for which the valid period and the periodicity are known. The valid period, like 'starting from September 1995 and ending by August 1998', shows the absolute time interval during which an association is valid, while the periodicity, like 'every weekend', conveys when and how often an association is repeated. Both the valid period and the periodicity are specified by calendar time expression in [Chen et al., 1998]. The cyclicity of association rules was similarly discussed in [Özden et al., 1998], where a cycle of an association rule was defined as a tuple such that the rule holds in every first time unit starting with the time unit. In [Ramaswamy et al., 1998], the concept of calendric association rules is defined, where the association rule is combined with a

calendar, such as 'all working days in 1998', that is basically a set of time intervals and is described by a calendar algebra.

Chen et al. [Chen et al., 1998] define temporal association rules according to the definition of temporal transactions, as follows:

> *Let* $I = \{i_1, i_2, \cdots, i_m\}$ *be a set of literals which are called items. A set of items* $X \subset I$ *is called an itemset. Let* D *be a set of time-stamped transactions over the time domain* T *which is a totally ordered set of time instants or chronons. Each time-stamped transaction* S *is a triplet* $\langle tid, itemset, timestamp \rangle$*, where* $S.tid$ *is the transaction identifier,* $S.itemset$ *is a set of items such that* $S.itemset \subseteq I$ *, and* $S.timestamp$ *is an instant (or chronon) which the transaction* S *is stamped with, such that* $S.timestamp \in T$ *. A transaction* S *contains an itemset* X *if* $X \subseteq S.itemset$ *[Chen et al., 1998].*

Furthermore, the definition of a temporal association rule is the form of:

> *Given a set of time-stamped transactions, a temporal association is a pair* $\langle AR, TF \rangle$ *, where* AR *is an implication of the form* $X \Rightarrow Y$ *and* TF *is a temporal feature that* AR *possesses. It expresses that during each interval* P *in* $\phi(TF)$*, the presence of* X *in a transaction implies the presence of* Y *in the same transaction [Chen et al., 1998].*
>
> - *AR has confidence c% during interval* P_i*,* $P_i \in \phi(TF)$*, if no less than c% of transactions in* $D(P_i)$ *that contain* X *also contain* Y*.*
>
> - *AR has support s% during interval* P_i*,* $P_i \in \phi(TF)$ *if no less than s% of transactions in* $D(P_i)$ *contain* $X \cup Y$*.*

- *AR possesses the temporal feature TF with the frequency f% in the transaction set D if it has minimum confidence min_c% and minimum support min_s% during no less than f% of intervals in* $\phi(TF)$.

In the above definition, the notion of frequency is introduced for measuring the proportion of the intervals, during which AR satisfies minimum support and minimum confidence, to the intervals in $\phi(TF)$. It is required that the frequency of any temporal association rule $\langle AR, TF \rangle$ should not be smaller than the user-specified minimum frequency which is a fraction within [0,1]. In the case that $\phi(TF)$ just includes a single interval, the frequency will be omitted since the meaningful minimum frequency must be 1 and AR must have minimum support and minimum confidence during this single interval [Chen et al., 1998].

Depending on the interpretation of the temporal feature TF, a temporal association rule $\langle AR, TF \rangle$ can be referred to as [Chen et al., 1998]:

- *A universal association rule if* $\phi(TF) = \{T\}$, *where T represents the time domain;*

- *An interval association rule if* $\phi(TF) = \{itvl\}$, *where* $itvl \subset T$ *is a specific time interval;*

- *A periodic association rule if* $\phi(TF) = \{p_1, p_2, \cdots, p_n\}$, *where* $p_i \subset T$ *is a periodic interval in cycles;*

- *A calendric association rule if* $\phi(TF) = \{cal_1, cal_2, \cdots, cal_m\}$, *where* $cal_j \subset T$ *is a calendric interval in a specific calendar.*

4.7.2 Mining Areas of Temporal Association Rules

With the above definition, the mining of temporal association rules has a two-dimensional solution space, that is, the space consisting of patterns and temporal features. According to different restrictions, the problems of mining temporal association rules can be classified into three groups as follows [Chen et al., 1998]:

- Finding temporal features of a given association rule (e.g., finding all valid time periods of a rule);

- Finding association rules with a given temporal feature (e.g., finding all rules that hold during the weekend);

- Finding all possible temporal association rules of a certain kind (e.g., interval association rules, periodic association rules, or calendric association rules).

Bases on the above definitions, Chen and Petrounias [Chen and Petrounias, 1999] [Chen and Petrounias, 2000a] pointed out in that, in many cases of applications, people might just be interested in some specific associations of some items in the database, but have no idea about when and/or how often these associations hold. The mining of temporal features of some specific association is a kind of meaningful and important problem in association rule discovery. These temporal features are [Chen and Petrounias, 1999] [Chen and Petrounias, 2000a]:

- Finding all interesting contiguous time intervals over a time domain during which a specific association holds.

- Finding all interesting periodicities that a specific association has.

- Validating a given temporal feature (e.g., a time interval, a periodicity, or a calendar) that a specific association has.

4.8 Summary

This chapter has formalised the problem of temporal data mining. The background of temporal data mining was presented and the definition of temporal data mining was given. Potential knowledge and patterns in temporal databases were pointed out, which include inducted pattern, trend pattern, similar pattern and sequential pattern. After that, temporal data mining operations were discussed in the categories of association rules, classification, clustering and prediction.

The temporal features and the representation of temporal features were introduced with some definition to support the following research works. Additionally, to formalise the research, the temporal association rule mining problem was defined and discussed in detail.

5 Discovering Interval Association Rules

5.1 Introduction

This chapter introduces an algorithm called IARMiner for discovering interval association rules. The problem background is pointed out, and the description of particular mining challenges that temporal association rules pose, follows. Thereafter, the related definition for mining interval association rules including that for interval association rule mining tasks, is provided to further formalise the mining problems. The actual algorithm IARMiner is clearly presented from association rules and longest interval searching techniques two aspects. Regarding implementation, a clear view of the programming for IARMiner is provided in every detail. Finally, an evaluation exercise is performed to examine the efficiency of IARMiner.

5.2 Problem Description

One interesting extension to association rules is the inclusion of a temporal dimension [Li et al., 1999] [Chen et al., 1998] [Chen, 1999]. For example, *bread* and *coffee* may be ordered together primarily between 7.00 am and 11.00 am. Therefore, the above association rule has support as high as 40% among the transactions that happen between 7.00 am and 11.00 am, and may have support as low as 0.005% among transactions other than those from 7.00 am to 11.00 am. As another example, if one interrogates a database of transactions in a supermarket, *turkey* and *pumpkin pie* are seldom sold together. However, if one only looks at the transactions in the week before Thanksgiving, most are discovered to contain *turkey* and *pumpkin pie*. That is, the association rule '*turkey* $\Rightarrow$ *pumpkin pie*' has high support and high confidence in the transactions occurring in the week before Thanksgiving. More examples of the differences between temporal association rules and non-temporal association rules are given in [Ale and Rossi, 2000] [Antunes and Oliveira, 1998] [Chen and Petrounias, 2000] [Chen and Petrounias, 2000a].

Also, if different time intervals are considered, different association rules can be discovered. Some association rules may hold during some time intervals but not others. Discovering temporal intervals as well as the association rules that hold during those time intervals may lead to useful information. Informally, the association rules along with their temporal intervals are referred to as temporal association rules [Li et al., 1999]. The usual objective is to discover regularities in the occurrence of certain events and temporal relationships between them [Ale and Rossi, 2000].

5.2.1 Previous Research Areas of Mining Temporal Association Rules

According to the classification of Chen et al. [Chen et al., 1998], the mining problem of temporal association rule includes: finding temporal features of a given association rule; finding association rules with a given temporal feature; and finding all possible temporal association rules of a certain kind.

Özden et al. [Özden et al., 1998] studied the problem of association rules that exist in certain time intervals and thus display regular cyclic variations over time. They present algorithms to efficiently discover what they called 'cyclic association rules'. It is assumed that time intervals are specified by the user. In both [Chen et al., 1998] and [Ramaswamy et al., 1998], the mining problem is limited to find all possible association rules with user-specified periodic or calendric features. That means to discover all rules with some known temporal features in which people might be interested.

In [Chen, 1999] and [Chen and Petrounias, 1998a], a framework for the discovery of temporal data mining was presented, and the discovery of the longest time intervals and the longest periodicities of association rules, were discussed. Because they believe that from a computational point of view, it is too expensive to find all possible hidden temporal association rules from large databases without any given restrictions, and in practice, the techniques for those restricted mining problems can be alternatively used in an interactive and iterative mining process in order to find all possible temporal association rules. The area concerned with finding all possible temporal association rules of a certain kind has not been touched.

As well as finding temporal features of given association rules and finding association rules with a given temporal feature, people also study temporal association rules from other viewpoints. Instead of studying the temporal features of when an association rule holds, Abraham and Roddick studied the temporal association rule problem by finding when an association rule changes [Abraham and Roddick, 1999]. Lee et al. made their focus on the temporal features of items in given datasets instead of association rules [Lee et al., 2001].

5.2.2 Challenge of Mining Interval Association Rules

Previous research like that of [Chen, 1999] and [Chen and Petrounias, 1998a] developed a number of techniques for discovering the temporal features of given associations or discovering associations with given temporal clues. Such research assumes that speculations can often be made by experienced experts and that those

speculations need only to be validated before being used for decision-making, but that is not always the case, because a real world application also requires that unimaginable temporal association rules must be discovered, without any known clue. Experts provide their speculations according to the knowledge they already possess, and data mining is a tool that helps them to discover that knowledge, so, experts can not always provide speculations before knowledge has been discovered. The unsolved challenge of temporal association rule mining techniques is to discover both temporal features and association rules without pre-given information.

The mining of interval association rules is one of the problems of mining temporal association rules which also includes mining periodic association rules; and the challenge comes from the expensive time and system consumption. The expenses of directly searching interval association rules are due to the multiple loop-repeat and regressive invocation when discovering both temporal features and association rules together. The multiple loop-repeat and regressive invocation is unavoidable because people can not treat both item and time information in the same way [Antunes and Oliveira, 1998]. This information must be treated separately, repeatedly and regressively. A typical association rule mining algorithm like Apriori makes the asymptotic time consumption of the complexity reach $f = \mathrm{O}(k \cdot 2^n)$, where k is the size of the dataset to be mined and n presents the quantity of items from the transaction dataset. For mining temporal features, such as longest time intervals, a typical algorithm like LISeeker [Chen, 1999] [Chen and Petrounias, 1998a] makes the asymptotic time consumption of the complexity reach $f = \mathrm{O}\left(k \cdot \frac{g^2 + 3g + 2}{6}\right)$, where k is the size of the dataset to be mined and g is the amount of granularity among the whole time domain. Both complexity notations were calculated in the worst situations. The worst situation for Apriori is that all candidate itemsets found by Apriori are frequent (satisfying the minimum support). And the worst situation for LISeeker is that the transaction dataset to be treated is spread equally into g granularity parts among the time domain, the minimum interval length threshold equals 1, and the given association rule holds on the whole time domain (the longest time interval equals the length of the time domain). Obviously, no one will simply combine two such algorithms together and regressively invoke Apriori and LISeeker to try to

discover both association rules and temporal features of all possible interval association rules.

5.3 Definitions Related to Mining Interval Association Rules

With respect to a given time-stamped dataset, there may be a series of different interval association rules existing among the time domain duration. The task of interval association rule mining is defined as follows:

Given a set of time-stamped transactions (D) over a time domain (T), minimum support (min_sup), minimum confidence (min_con), and minimum interval length (min_len), the problem of mining interval association rule (IAR) is to find all possible association rules (ARs) and all their corresponding longest time intervals during which the association rules hold.

5.3.1 Temporal Features Related to the Interval Association Rule Mining Task

To model the interval association rule mining problem, the issue of time intervals must be introduced firstly, in every respect. Chen and Petrounias [Chen, 1999] [Chen and Petrounias, 1998] [Chen and Petrounias, 1999] [Chen and Petrounias, 2000a] [Chen et al., 1998] gave a number of precise definitions about them.

- ***Interval***:

  Assume that each interesting time interval is composed of a totally ordered set of contiguous constructive intervals with a given granularity. Here, such a constructive interval is called a granular interval that is a non-decomposable interval of some fixed duration. The interval granular is the size of each granular interval (e.g., Hour, Day, Week, Month, etc.) which may vary with different applications and user's interests.

➢ ***Valid Interval***:

An interval (ITVL) is valid with respect to an association AR if the temporal association rule $(AR, ITVL)$ *satisfies min_sup (minimum support) and min_con (minimum confidence) during that interval.*

More often than not people are only interested in those intervals that are long enough, since some short intervals may not be periods of particular interest or concern for a particular application. Under the requirement of different applications, the users can specify a minimum length (*min_len*) of the expected intervals. According to this minimum length, the concept of long intervals can be defined as follows [Chen, 1999] [Chen and Petrounias, 1998] [Chen and Petrounias, 1999] [Chen and Petrounias, 2000a] [Chen et al., 1998]:

➢ ***Long interval***:

Given a minimum interval length and an interval granularity (GC), an interval (ITVL) is long with respect to an association rule (AR) if:

1) *ITVL is valid with respect to AR, and*
2) $length(ITVL, GC) \geq min\_len$. *(minimum interval length).*

➢ ***Strictly long***:

Given a minimum interval length and an interval granularity GC, an interval ITVL is strictly long with respect to association AR if for any ITVL', $ITVL' \subset ITVL$ *and* $length(ITVL', GC) \geq min\_len$, *ITVL' is long with respect to AR. With respect to a given association AR, for any two strictly long intervals,* $ITVL_1$ *and* $ITVL_2$, *if* $ITVL_1 \subset ITVL_2$, *we say that* $ITVL_2$ *is strictly longer than* $ITVL_1$.

➢ ***Longest interval***:

Given a minimum interval length and an interval granularity GC, an interval ITVL is longest with respect to an association rule AR if:

1) *the interval ITVL is strictly long with respect to AR, and*

2) *there is no strictly long interval ITVL'' with respect to AR, such that* $ITVL'' \supset ITVL$.

5.3.2 Additional Definition for Mining Interval Association Rules

With respect to the definitions above, which are used in the algorithm of mining the longest interval of a given association rule (LISeeker [Chen and Petrounias, 1999b]), the definition of an interval association rule is:

> Given a set of time-stamped transactions, an interval association rule is a pair $\langle AR, ITVL \rangle$, where AR is an implication of the form $X \Rightarrow Y$ and $ITVL$ is a time interval during which the AR holds. It expresses that during each interval P in the time domain, the presence of X in a transaction implies the presence of Y in the same transaction. So:
>
> 1) AR has confidence $c\%$ during interval P, if no less than $c\%$ of transactions in $D(P)$ that contain X also contain Y.
> 2) AR has support $s\%$ during interval P, if no less than $s\%$ of transactions in $D(P)$ contain $X \cup Y$.

Obviously, one can say two interval association rules are the same, if and only if they have the same AR and the same time interval.

5.4 Algorithm IARMiner

The algorithm IARMiner (Interval Association Rule Miner) was developed to discover interval association rules (*IARs*). Given a dataset with a time stamp for each transaction, a minimum support (*min_sup*) and a minimum confidence (*min_con*), and even more a basic time granularity unit (*GC*) and a minimum length of interval (*min_len*), the discovery of interval association rules is to find out all possible association rules (*ARs*) and the corresponding longest time intervals (*LIs*) on which those association rules hold. The process of finding interval association rules falls into two aspects, finding association rules and finding their corresponding longest time

intervals. From another point of view, the focus is on finding all possible association rules which satisfy the given minimum support and minimum confidence during the longest intervals, which are not less than the given minimum interval length. One association rule may exist during the whole time domain or just part of it, and moreover, can exist during some intervals within the time domain regularly or irregularly.

5.4.1 Longest Interval Searching Technique

Suppose there is a set of time-stamped transactions D which are ordered by timestamps, and the time domain $T = \{G_1, G_2, \ldots, G_n\}$, where $G_i \, (1 \le i \le n)$ is a granular interval. The data set D can be partitioned into $\{D(G_1), D(G_2), \ldots, D(G_n)\}$. The problem of searching interval association rules can be considered as successively looking for all possible association rules (ARs) and their longest sequences $\{G_1, G_2, \ldots, G_n\}$ among the time domain, during which the AR holds. The whole searching process can be performed in two dimensions: finding all ARs, and finding all their corresponding longest intervals.

As Figure 5.1 shows, there can be a number of $IARs$ existing in the time domain, IAR_1 and IAR_3 may have the same AR but different time intervals, and likewise IAR_1 and IAR_5 may have the same time interval but different AR.

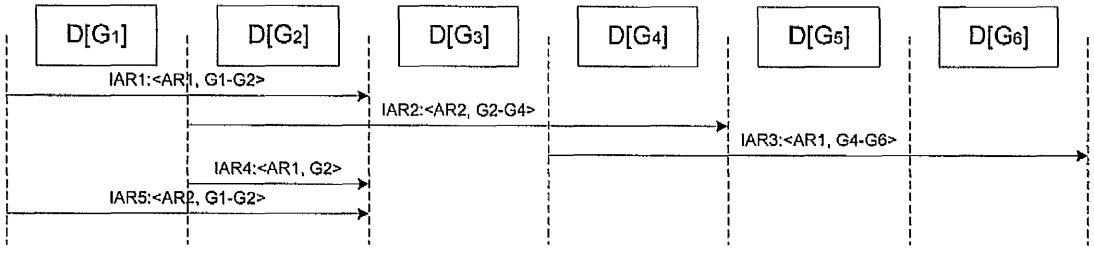


Figure 5.1 Interval association rules

As many algorithms like Apriori have been developed to mine association rules, the remaining problem is how to find the corresponding longest intervals when mining association rules. A method called interval combination is introduced to fulfil such

requirement. Figure 5.2 shows the IAR_1 on the interval of $itvl_1$. As the definition of *IAR* mining, the interval during which the *AR* holds has to be strictly long. According to the definition of 'strictly long' any sub-interval longer than *min_len* has to be strictly long also, meaning that, *AR* must hold on $itvl_2$, $itvl_3$ and $itvl_4$.

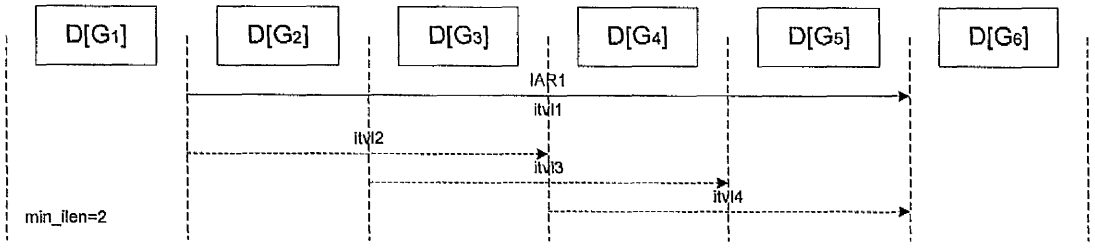


Figure 5.2 Strictly long interval

Searching for longest intervals follows the opposite way. That means that if the mining process finds that an *AR* holds on $itvl_1$ and $itvl_2$ (Figure 5.3) and they are both strictly long, one can combine them together to get a longer interval $itvl_3$. If the *AR* still holds on $itvl_3$, the $itvl_3$ is also a strictly long interval. It is not necessary to test the hold of the *AR* on any sub-interval of $itvl_3$. Likewise, if $itvl_4$ and $itvl_5$ have the same *AR* and are both strictly long, the condition of $itvl_6$ being strictly long is only that the *AR* holds on $itvl_6$ (instead of holding on $itvl_4$, $itvl_5$ and $itvl_6$). But one can not simply combine $itvl_7$ and $itvl_9$ to get and verify $itvl_{10}$, because according to the definition of *IAR*, the $itvl_8$, as a sub-interval of $itvl_{10}$, is longer than the minimum length and must be strictly long, which can not be conformed.

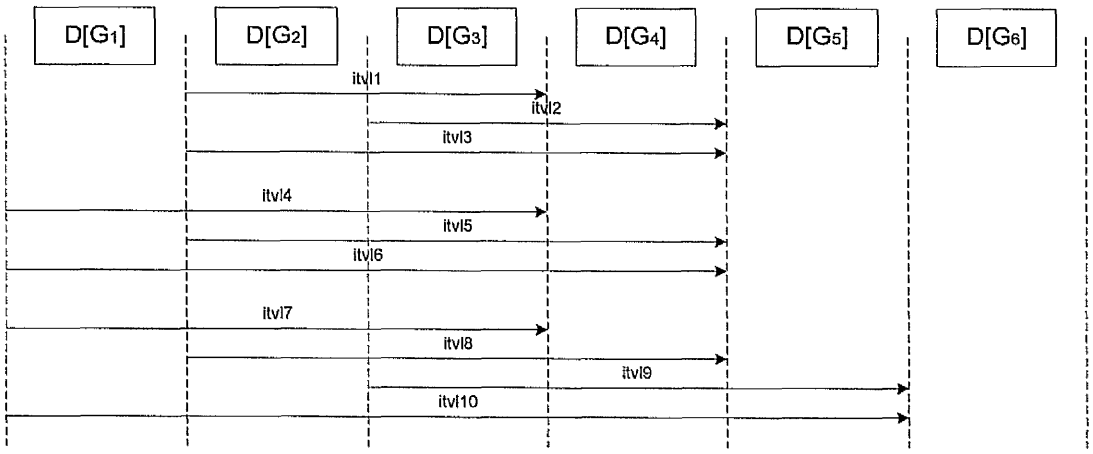


Figure 5.3 Interval combination

Definition of combinable intervals:

Combinable intervals (CI):

Given two strictly long time intervals $itvl_1$ and $itvl_2$, and a granularity GC, one can say that $itvl_1$ and $itvl_2$ are combinable if

1) $length(itvl_1, \ GC) = length(itvl_2, \ GC)$, and
2) the difference between two start time granularities (or end time granularities) of $itvl_1$ and $itvl_2$ equals one granularity.

To find the longest intervals of a given association rule, the algorithm IARMiner takes the procedure as: finding the minimum length intervals that the association rule (AR) holds on, combining each pair of combinable intervals (CIs) to get longer intervals whose lengths are min_len+1, verifying the combined intervals to still be long, and repeating the combining process until the longest time intervals are found.

Figure 5.4 shows the basic procedure of searching the *IARs* in IARMiner. Firstly, every interval equalling the minimum interval length has to be searched to find all possible *ARs*, as shown in step 1 in Figure 5.4. Secondly, for every pair of combinable intervals, if they have the same AR part, they should be combined to get a longer interval, and verified as still being long. The same process continues until any of the following three conditions are reached:

1) no more combinable intervals

2) no more combinable intervals have the same AR (or ARs)

3) the length of the longest interval already reaches the whole time domain.

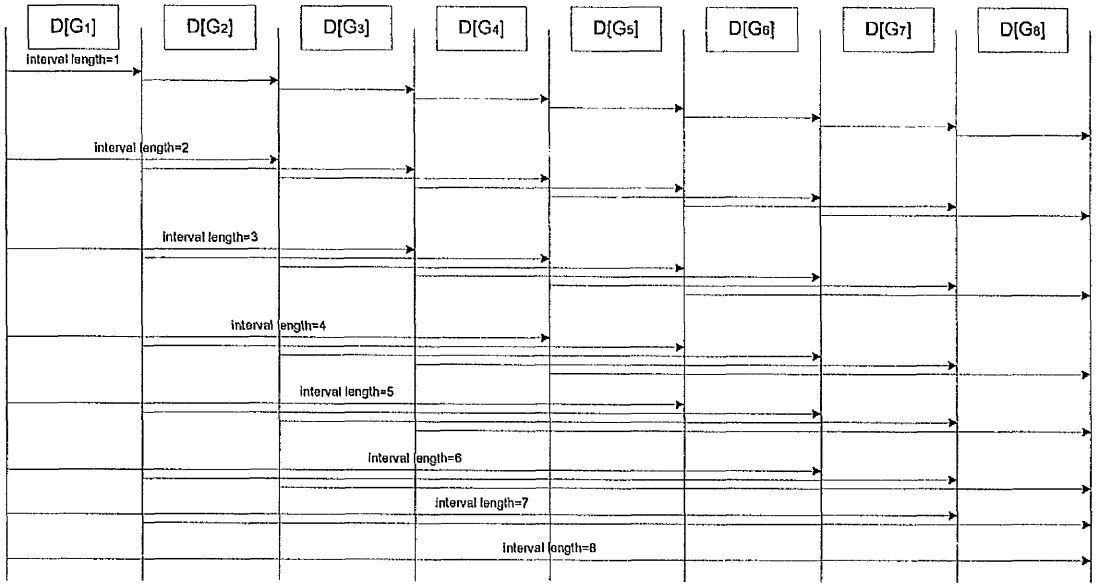


Figure 5.4 Searching interval association rule

5.4.2 Association Rule Searching Technique

There is no special sub-algorithm of IARMiner for searching association rules within each minimum length interval from the temporal datasets. Most common association rule mining algorithms can be used in IARMiner. Apriori [Agrawal and Srikant, 1994] is chosen for this research because of its simplicity, principles, typicality and understandability. In IARMiner, Apriori is used to mine all ARs from every possible minimum length sub-dataset. The job of verifying combined intervals as still long is performed by scanning and counting the AR part(s) on the sub-datasets of the new interval length.

5.4.3 Algorithm Description

This section describes the algorithm IARMiner for searching all interval association rules ($IARs$) in a database D over a time domain T. For simplicity and without loss of the granularity, suppose $T = \{G_1, G_2, \ldots, G_n\}$ and $D = \{D[G_1], D[G_2], \ldots, D[G_n]\}$.

The searching process is gradually made by scanning all partitions of the database $D[G_1], D[G_2], \ldots, D[G_n]$.

For monitoring the searching process a list data structure, IL (interval list) is introduced to help solve the problem. Every node of IL contains three parts (Figure 5.5):

$Start\_G$: start granularity (start time point);

End_G : end granularity (end time point);

$ARset$: a set of association rules holding on the interval that starts with $Start\_G$ and ends with End_G.

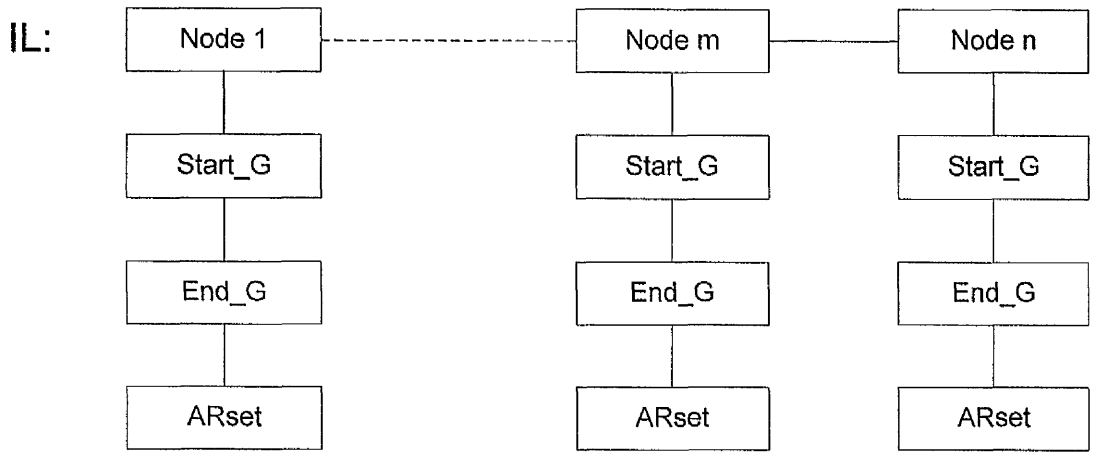


Figure 5.5 Data structure of interval list

To help combine the possible combinable intervals, two pointers are used to represent the nodes of IL, $Lptr_1$ and $Lptr_2$ (Figure 5.6).

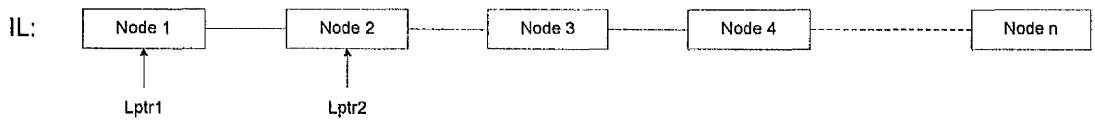


Figure 5.6 Pointers used for interval list

The first step of IARMiner is to build the IL list by searching all AR s on every sub-dataset whose interval equals the minimum interval length. After that, the searching process falls into the loops of combining each pair of combinable intervals having the

same $AR(s)$ and trying to get and verify them. There is only one condition to finish the mining process, that is where only one, or no node is left in the IL after the combination loops, but three conditions apply in order to break a loop:

1) the current and the next list nodes have at least two different granularities, ($|start\_G_1 - start\_G_2| \geq 2$), (not combinable)
2) the two IL nodes to be combined have no same AR in their $ARsets$, ($ARset_1 \cap ARset_2 = \varnothing$)
3) the new combined interval is no longer long (candidate AR s no more hold).

In addition, some functions are designed for manipulating IL as follows:

$Output(ARs)$: output the IAR;

$Scan(D[G_m],\ldots,D[G_n])$: scan the dataset partitions from $D[G_m]$ to $D[G_n]$ to find out whether there are AR (s) holding on such duration (interval).

```
Data structure : List
Dataset length = n
Every list node contains : Statt granular : Start_G
                           End granular : End_G
                           a set of ARs (Frequent Itemset)
First step :
ptr1 = 1
ptr2 = ptr1 + min_ilen - 1
while(ptr2 ≤ n)do
{   G_Start = ptr1;
    G_End = ptr2;
    Input dataset from G_Start to G_End;
    Scan dataset for ARs (Frequent Itemset);
    if found
      IL Add(Strat_G = ptr1, End_G = ptr2, ARset = ARs);
    ptr1 += 1;
    ptr2 += 1;
}
```

```
Second step :
lptr1 = IL.first( );
lptr2 = lptr1.next( );
while(IL.length ≥ 2)do
{    if(lptr2.Start_G − 1 > lptr1.Start_G)
       then{output(lptr1.ARset);
            lptr1.remove( );
            lptr1 = lptr2;
            lptr2 = lptr2.next( );
            continue;
            }
     if(lptr1.ARset ∩ lptr2.ARset = ∅)
       then{output(lptr1.ARset);
            lptr1.remove( );
            lptr1 = lptr2;
            lptr2 = lptr2.next( );
            continue;
            }
     for all(ARs = lptr1.ARset ∩ lptr2.ARset)do
            {scan dataset from .lptr1.Start_G to lptr2.End_G
                 for ARs that still hold on the longer interval };
     if(new ITVL found(or : ARs ≠ ∅))
       then{output(lptr1.ARset − ARs);
            lptr1.ARset = ARs;
            lptr1.End_G+ = 1;
            lptr1 = lptr2;
            lptr2 = lptr2.next( );
            continue;
            }
       else{ output(lptr1.ARset);
            lptr1.remove( );
            lptr1 = lptr2;
            lptr2 = lptr2.next( );
            continue;
            }
 }// End of while
if(IL.length = 1)
  then output(lptr1.ARset);
```

5.5 Implementation

The algorithm IARMiner is realised using the Java programming language in the development environment of Microsoft Visual J++ 6.0 on a computer running WindowsXP operating system.

5.5.1 Program Structure - Overall Structure

The whole program is supposed to run the algorithm IARMiner as a sequence of input-output (Figure 5.7). After the program begins, it needs the input from the user in order to specify the target temporal transaction database file stored on a hard disk, as well as the minimum support, minimum confidence and minimum interval length thresholds. The target temporal transaction dataset file keeps the transactions in plain text in the format of one transaction to one line. Minimum support and minimum confidence thresholds are collected from user inputs in the form of a percentage for easy understanding. The minimum interval length threshold can be specified by the user as the multiple of a granularity unit, which is set to 'day'. The reason that this implementation does not accept specification of the granularity unit is due to the fact that the program will only be used for evaluation purposes, and the target transaction dataset was collected at the unit of day.

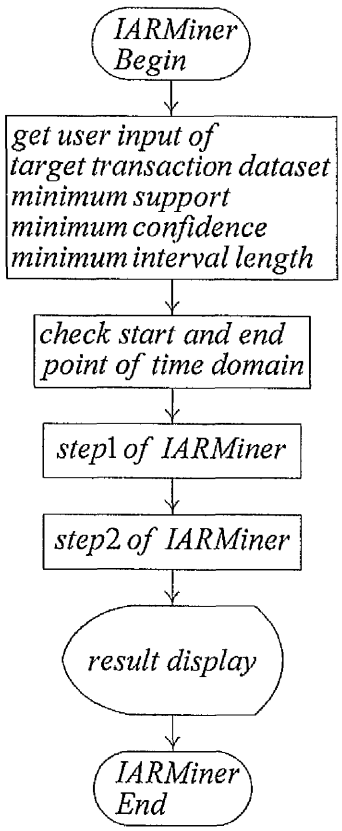


Figure 5.7 Program structure of IARMiner

After collecting the necessary information from the user, the program will then check the target temporal transaction dataset firstly to determine the start and end point of the time domain. These points will be used to control the loops of the following processes.

Once all the necessary information has been obtained, the program automatically starts the mining process, which is separated into two parts, which make it much easier to control and program, With respect to the algorithm IARMiner, the program IARMiner also performs the association rule mining job on each minimum length interval in step 1. Then step 2 finishes the job of combining each short interval association rule to the longest intervals.

Before ending the program, the final task is to display the resulting interval association rules and the system response time. For easy collection of the time consumption reading, the program IARMiner collects the computer system clock time

before the beginning of step 1 and just after the end of step 2. The time consumption measurements coming from the difference of these two system clock times are more accurate than a wall clock. But, because the platform of evaluation is a computer running the Windows operating system, the response time for the same inputs may get different results, which are due to some unpredictable processes running at the back of operating system. This can be reduced by minimising all unnecessary processes and repeating the evaluation to obtain average readings.

5.5.2 Program Structure - Step 1 of IARMiner

Because step 1 of IARMiner finishes the process of mining association rules on each minimum length interval, the most important element of the implementation of the step 1 is controlling the loading of the required transaction dataset for each minimum length interval, and realising the chosen regular association rule mining algorithm, which is Apriori [Agrawal and Srikant, 1994] in this research.

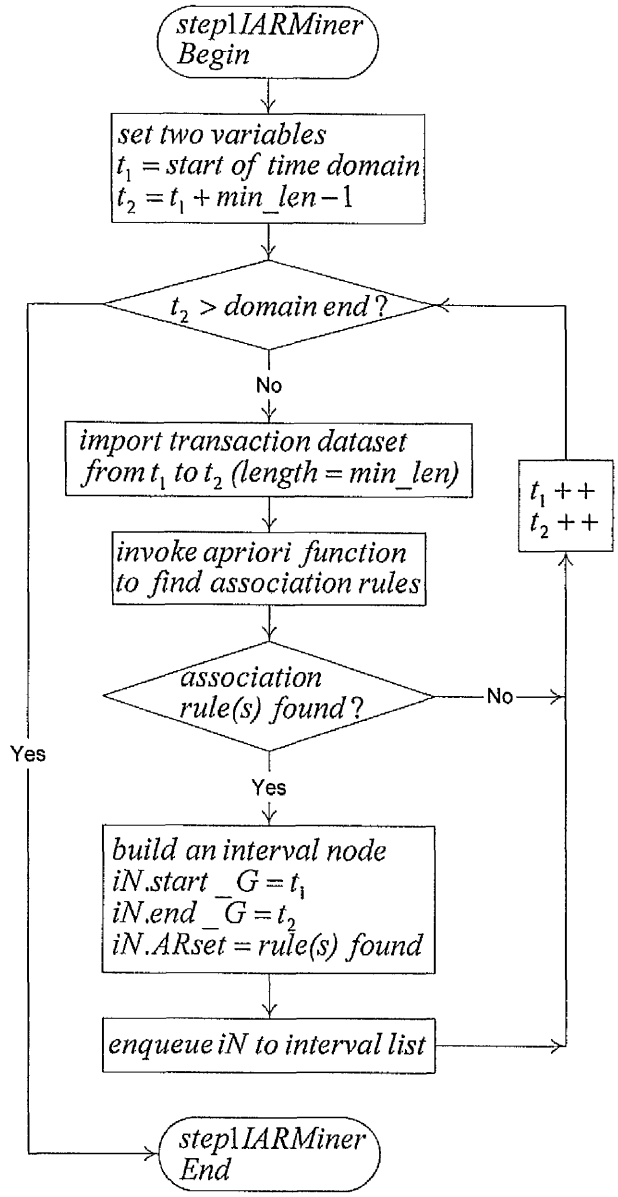


Figure 5.8 Program structure of step 1 of IARMiner

As Figure 5.8 shows, the program uses two variables to control accurate loading of transaction datasets on each minimum length interval for the purpose of regular association rule mining. In each loop, two variables t_1 and t_2 keep the same distance of minimum interval length. And t_2 will also be used to determine whether the whole time domain has been searched. The interval node will only be built while association rule(s) is/are actually found at corresponding minimum length intervals to reduce system consumption.

5.5.3 Program Structure - Step 2 of IARMiner

Step 2 of the IARMiner program finishes the process of combining short combinable interval association rules for the longest one (Figure 5.9). Two pointers iN_1 and iN_2 are used to reference two directly-linked interval nodes, which may be combinable. Because any two interval nodes which are not linked directly in the interval list must have at least two granularity differences between the start granularity point ($iN_2.start\_G - iN_1.start\_G \geq 2$), they are definitely not combinable. New interval nodes are linked to the end of the interval list when new longer interval association rules are being found. Since the two pointers iN_1 and iN_2 keep traversing down this growing interval list, all longest interval association rules will be found while the pointers reach the end of the list.

In programming, three procedures are introduced to determine whether two directly-linked interval nodes have combinable interval association rules. Firstly, the interval length of the two interval nodes are checked to see if they have the same length. Secondly, if two interval nodes have the same length, the difference between them will be checked by comparing their start granularities ($start\_G$). Finally, if they satisfy the above two conditions, the intersections of the two association rule sets ($ARsets$) will be checked to determine whether they have the same association rule(s).

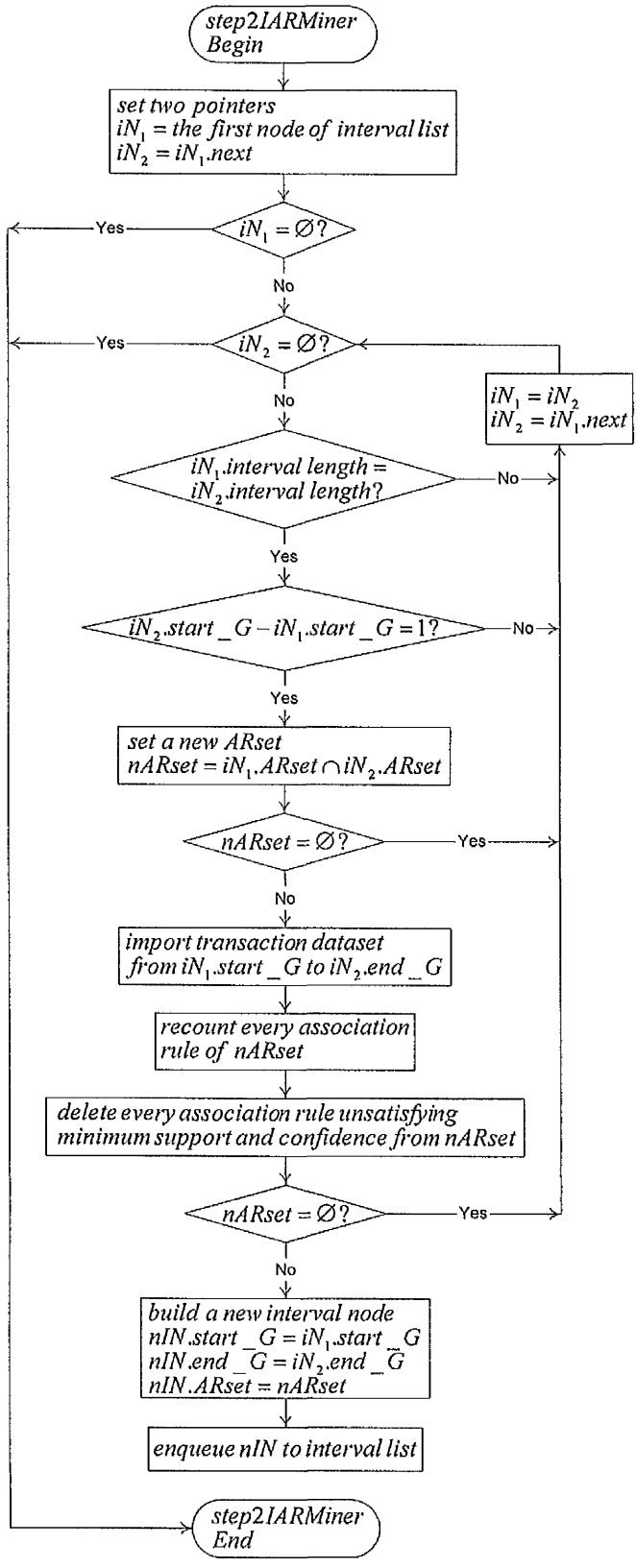


Figure 5.9 Program structure of step 2 of IARMiner

The actual program code is included in appendix-a.

5.6 Evaluation

To evaluate the performance of this interval association rule mining algorithm, a set of experiments were conducted to quantify and measure the effectiveness of the actual algorithm. This algorithm IARMiner was implemented using the Java programming language on the platform of a laptop computer running Microsoft Windows XP Professional operating system with Microsoft Visual J++6.0 programming environment. The laptop was equipped with an Intel PentiumM 1.5GHz CPU, 60GB hard drive and 1GB memory.

5.6.1 Experiment Dataset

A set of synthetic datasets and a real business world dataset were used to evaluate the effectiveness and efficiency of the algorithm.

The synthetic dataset

The synthetic dataset used for the evaluation was designed to meet the requirements of a temporal transaction dataset, fulfilling the coverage of a one-year time period, setting the amount of items to 50 for limiting the evaluation time, and setting the most association frequent itemsets to reach 50% support. In the synthetic dataset, there were 14,637 transactions, with an average of 10 items in each transaction.

The real business dataset

To overcome the shortcoming of the synthetic dataset, which is over-concentration of itemsets' support (to 50%), a real world temporal transaction dataset was introduced in the evaluation. This dataset came from a retail company, whose business was selling office copy machines, supporting parts and other office products. The transaction dataset covers the period from January 2003 to December 2005 - 36 months totally with 140,702 transactions containing 9,605 product IDs (item

number). After categorising, there were 152 new categories (new item number) and 95,138 transactions remained for evaluation.

5.6.2 Time Consumption Experiments

The response time was measured as the time that elapsed from the initiation of the execution of the actual mining process to the end time of finishing the computation. Experiments were repeated 5 times to obtain stable values for each data point.

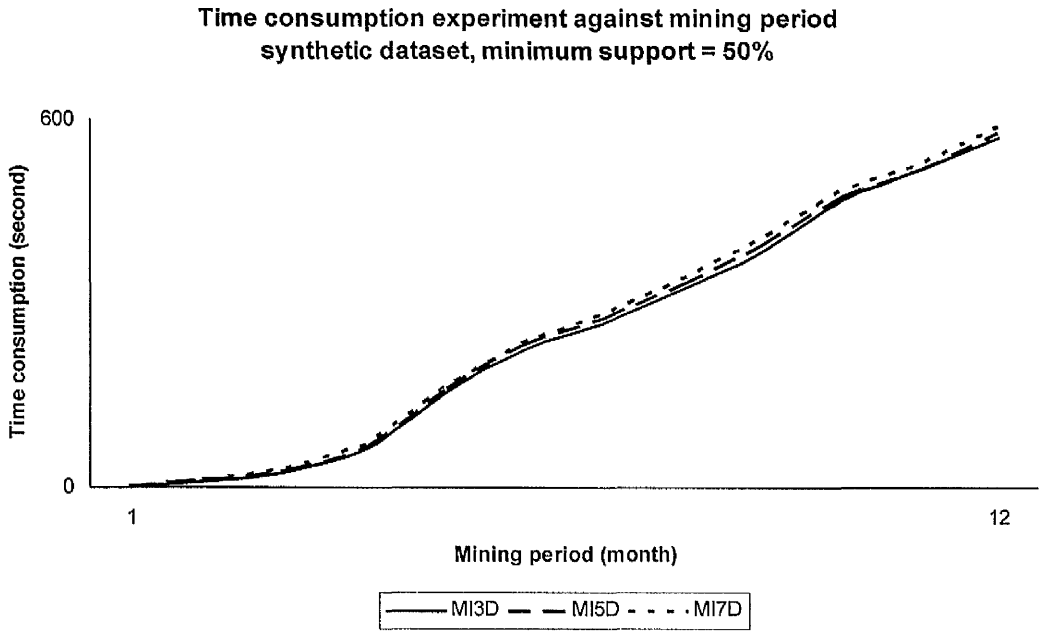


Figure 5.10 Time consumption of IARMiner

Figure 5.10 shows the time consumption experiment against the mining duration on the synthetic dataset with the minimum support equalling 50%, on which most frequent itemsets hold. The response time was measured as the time which elapsed from the start of loading the first dataset to the end time of finishing the computation. The experiment collected 144 response times from the 144 tests based on different mining periods and different minimum interval thresholds. For a clear view, Figure 5.10 shows only 36 response times in three groups which differ from each other on minimum interval length thresholds, represented in the form of *MIxD*. In the form, *x* is the minimum interval length in the granular unit of 'day'. Although for a clear view

only the experiment results of 3 days, 5 days and 7 days of minimum interval length thresholds are shown on the chart, like these three groups, all other results, whose used minimum interval length vary from 1 to 23 , showed the same trend of reasonable response time for the mining duration. The reason for the different response time given by different minimum interval lengths was developed in the next experiment.

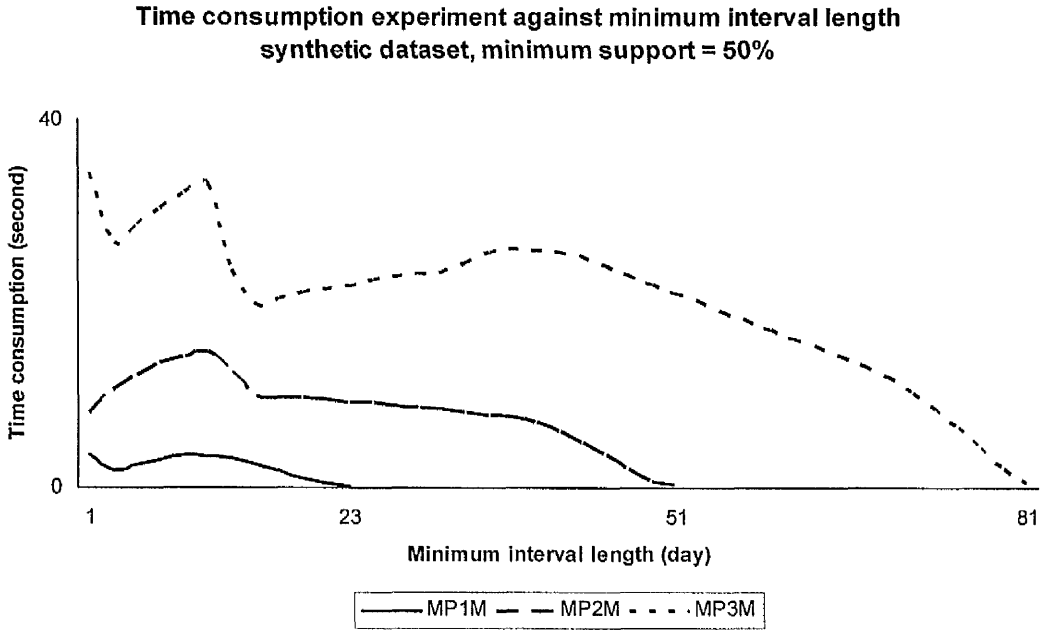


Figure 5.11 Time consumption of IARMiner

Figure 5.11 shows the result of the time consumption experiment against minimum interval length on the synthetic dataset with the minimum support threshold equalling 50%. The experiment collected 65 response times on different mining durations and different minimum interval length thresholds. The chart (Figure 5.11) shows only the response time in three groups which differ from each other on mining duration length, represented in the form of *MPxM* . In the form, x is the mining duration in the unit of 'month'. The results pointed out an interesting phenomenon, that being, the response time rises while the minimum interval length threshold approaches half the length of the mining duration, and dramatically drops when the minimum interval length is longer than half the length of the mining duration and reaches the whole length of the mining duration. (After calculation, it is clear that the undulation of the experiment

results during only small minimum interval lengths are due to the uneven density distribution of the synthetic dataset, and can be seen as noise.)

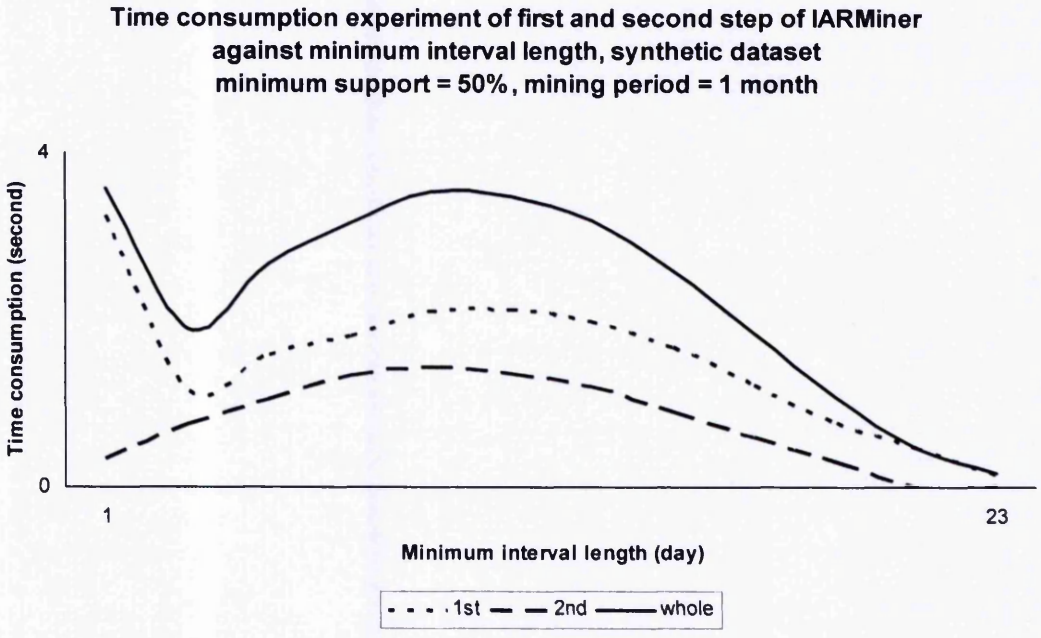


Figure 5.12 Time consumption of IARMiner

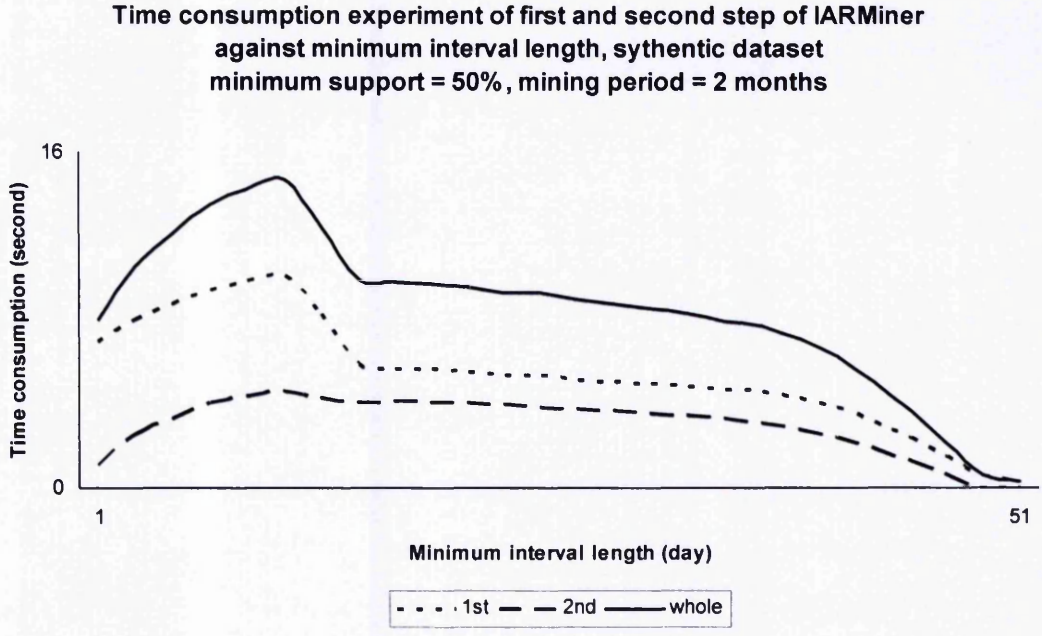


Figure 5.13 Time consumption of IARMiner

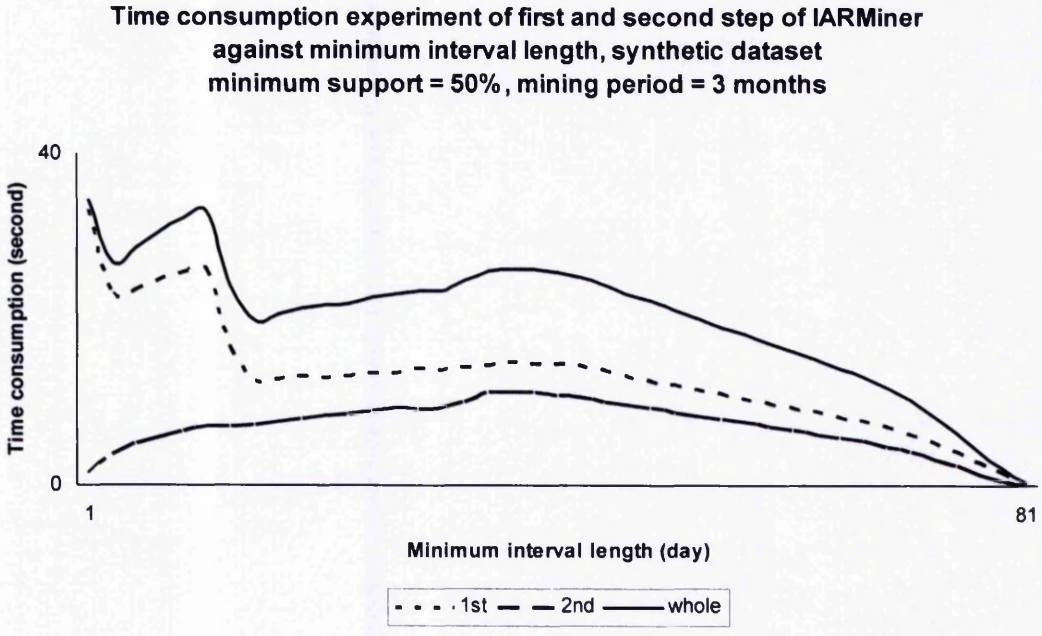


Figure 5.14 Time consumption of IARMiner

To study this phenomenon, three more experiments were performed on the same synthetic dataset but using different mining durations (Figure 5.12, Figure 5.13 and Figure 5.14). In these experiments the response time was collected after the end of the first mining step of IARMiner as well as the second. Three experiments ran on one-month (Figure 5.12), two-month (Figure 5.13), and three-month (Figure 5.14) mining durations. All of them show that the first mining step of IARMiner takes more time than the second, and plays the leading role in time consumption experiments of IARMiner.

This dramatic phenomenon occurs because of IARMiner's relatively expensive time consumption during the first running step, and the fact that the minimum interval length threshold reaches half the length of the whole mining duration. Figure 5.15 simulates the asymptotic time consumption result of the first step of IARMiner against different mining durations. The time consumption of the first step of IARMiner reaches the highest level while the minimum interval length approaches half the length of the total mining time domain. And the worst situation occurs when the minimum interval length equals half of the total mining time domain length.

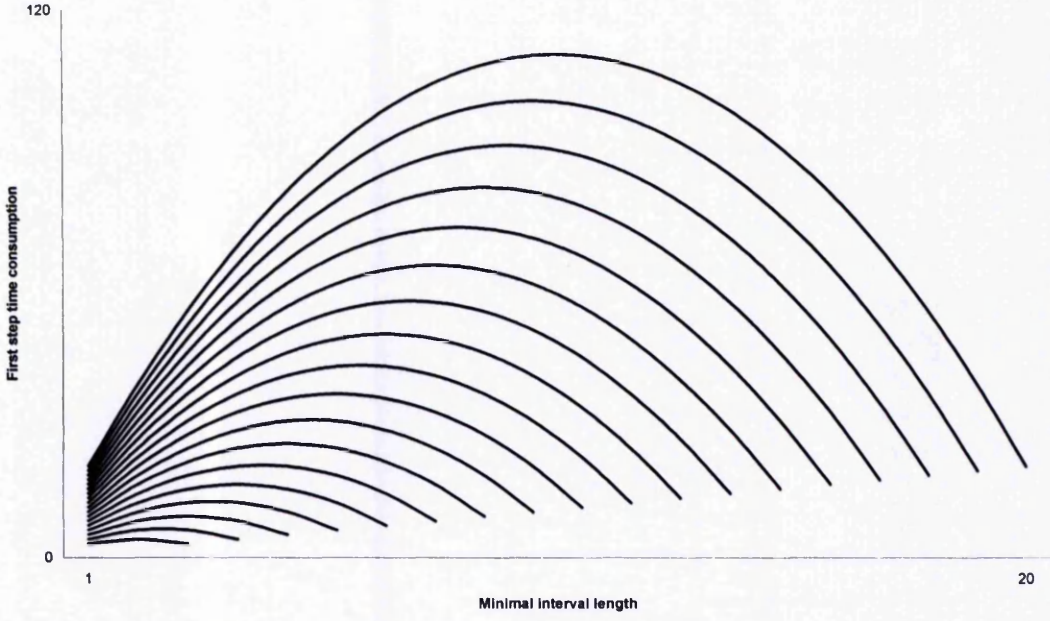


Figure 5.15 Time simulation of 1st step of IARMiner

Figure 5.15 was drawn based on the asymptotic time complexity of the first step of IARMiner. As Figure 5.16 shows, while the time domain length is 5, the total granularities to be mined during the first step of IARMiner increase from 5 to 9 when the minimum interval length increases from 1 to 3. It also reduces from 9 to 5 while the minimum interval length keeps rising.

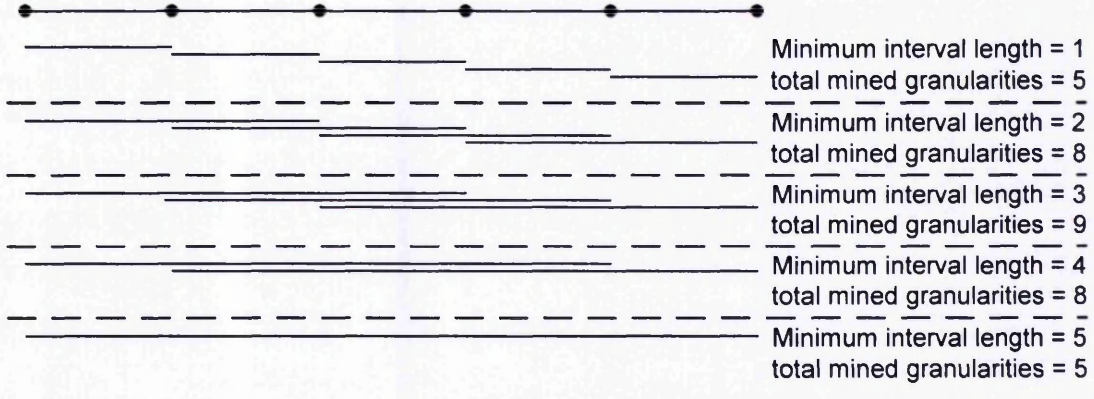


Figure 5.16 Granularity simulation of 1st step of IARMiner

After inducing, the asymptotic time complexity of the first step of IARMiner is as follows: Suppose that, the length of the time domain to be mined is n and the minimum interval length threshold is x, so, while $x \leq n$, the total temporal datasets in

the unit of granularity to be mined by the first step of IARMiner will be $y = x \cdot (n+1) - x^2$. Then, when the total time domain to be mined has three granularities ($n = 3$), the asymptotic time complexity of the first step of IARMiner will be $f = O(k \cdot (4x - x^2))$, as well as when $n = 4$ the complexity is $f = O(k \cdot (5x - x^2))$ and when $n = 5$, the complexity is $f = O(k \cdot (6x - x^2))$. The variable k is the time complexity of the chosen regular association rule mining algorithm, which is $f = O(2^n)$ here as Apriori is chosen as the regular association rule mining algorithm in this research.

In conclusion, the time consumption of the first step of IARMiner is predictable while the time consumption of the second step is really based on the distribution of interval association rules. Because in a real world application, the mining durations are always much longer than the user-specified minimum interval length, the worst situation rarely has the opportunity to occur.

The following experiments were performed on the real business temporal transaction dataset, which provides IARMiner with a more suitable evaluation while using it in the real world. The minimum support was set to 15% , at which most interval association rules were found. The experiments were repeated when the minimum interval length threshold was set to three, five, and seven days to give a comparison.

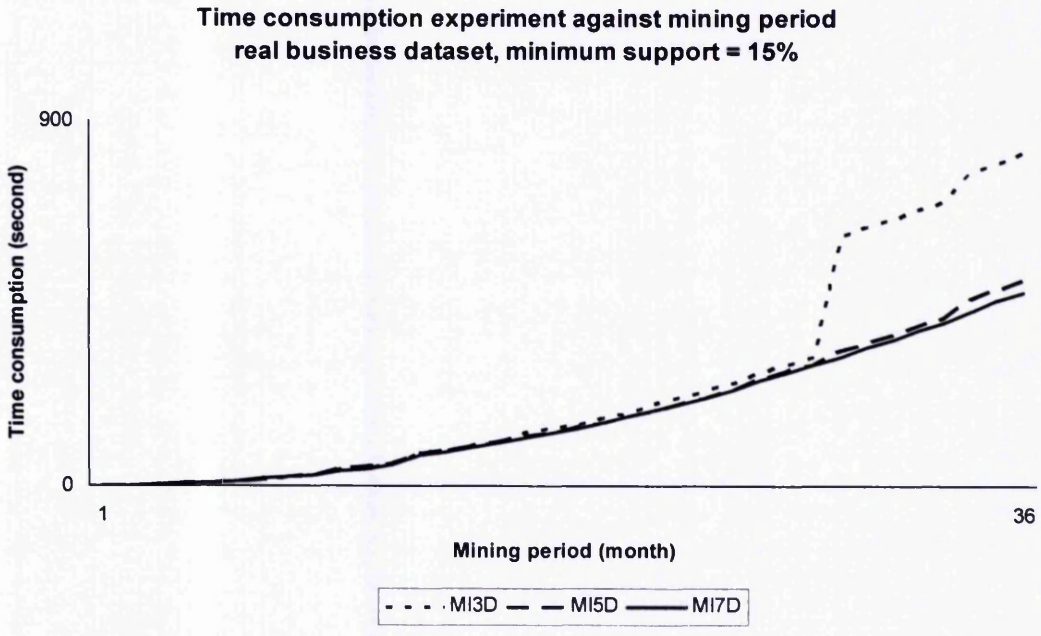


Figure 5.17 Time consumption of IARMiner

Figure 5.17 shows the time consumption experiment against mining durations on the real business dataset with the minimum support threshold equal to 15%, at which most interval association rules hold. The evaluation mining duration was 36 months, and the experiment was repeated three times with the minimum interval lengths equalled three, five, and seven days, which is presented in the chart in the form of *MIxD*. While the minimum interval length was three days, the response time raised above the average level at the duration approaching the 36*th* month. The reason is that, more $length-3$ interval association rules were found during that period.

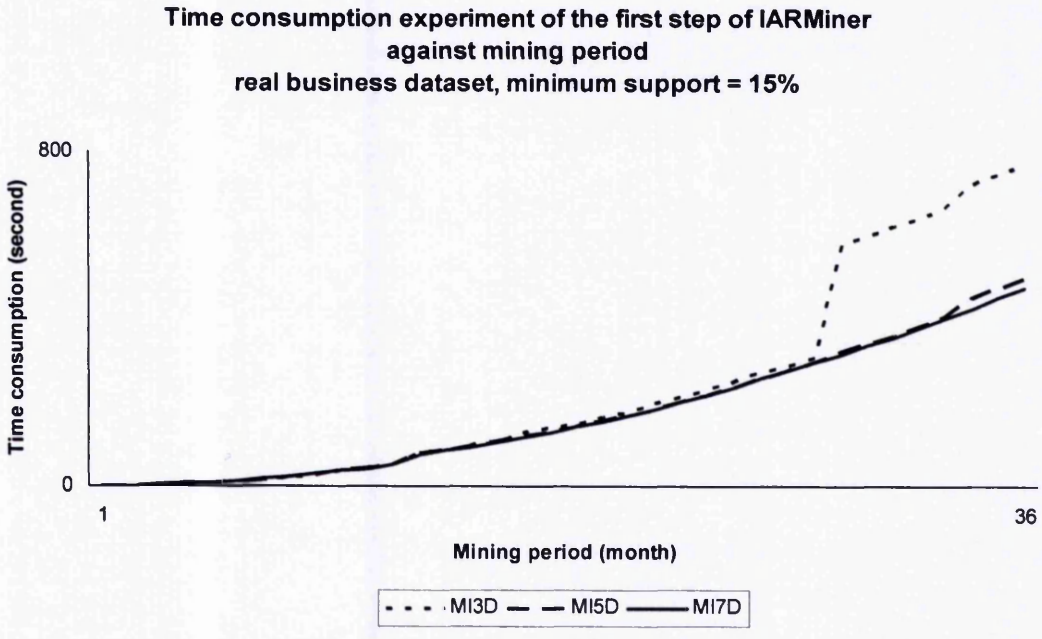


Figure 5.18 Time consumption of 1st step of IARMiner

To determine which procedure took more time consumption in IARMiner, two charts are given to show the individual time consumption of the first and second steps of the program. From Figure 5.18, it can be seen that the chart shape of the time consumption experiment results of the first step of IARMiner has a very similar shape to the consumption of the whole program (Figure 5.17). Comparing this to the following Figure 5.19 of the second step of IARMiner, it can be seen that for a real business dataset, the whole time consumption of IARMiner is decided by its first step. The second step only slightly increases the time consumption when more candidate minimum length intervals are found in the first step.

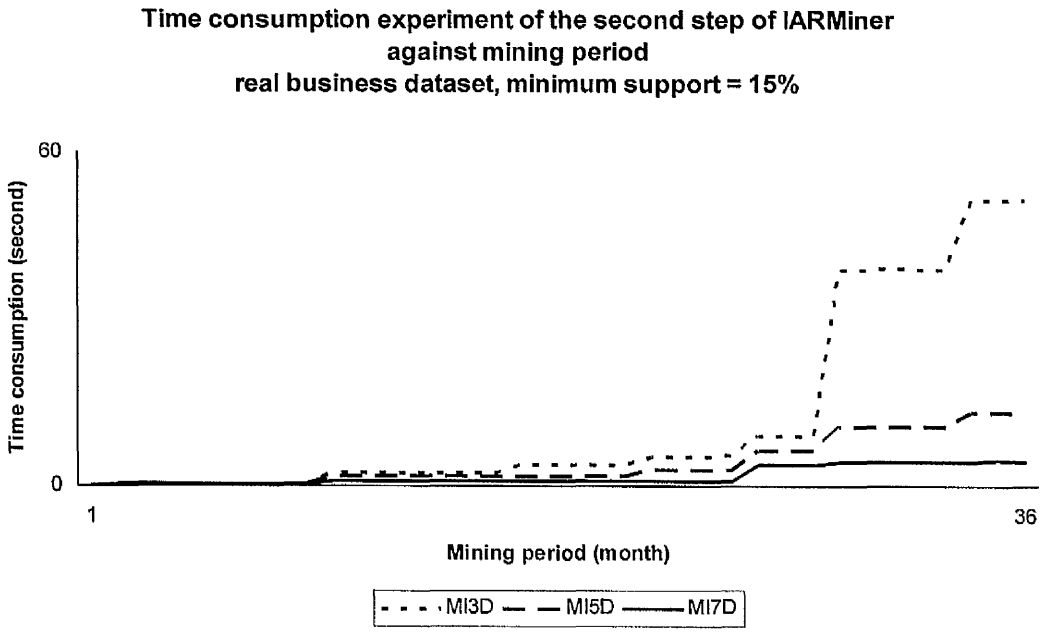


Figure 5.19 Time consumption of 2nd step of IARMiner

Furthermore, additional experiments were performed to indicate that. These experiments put the first and second steps of IARMiner in the same charts to provide a clearer view.

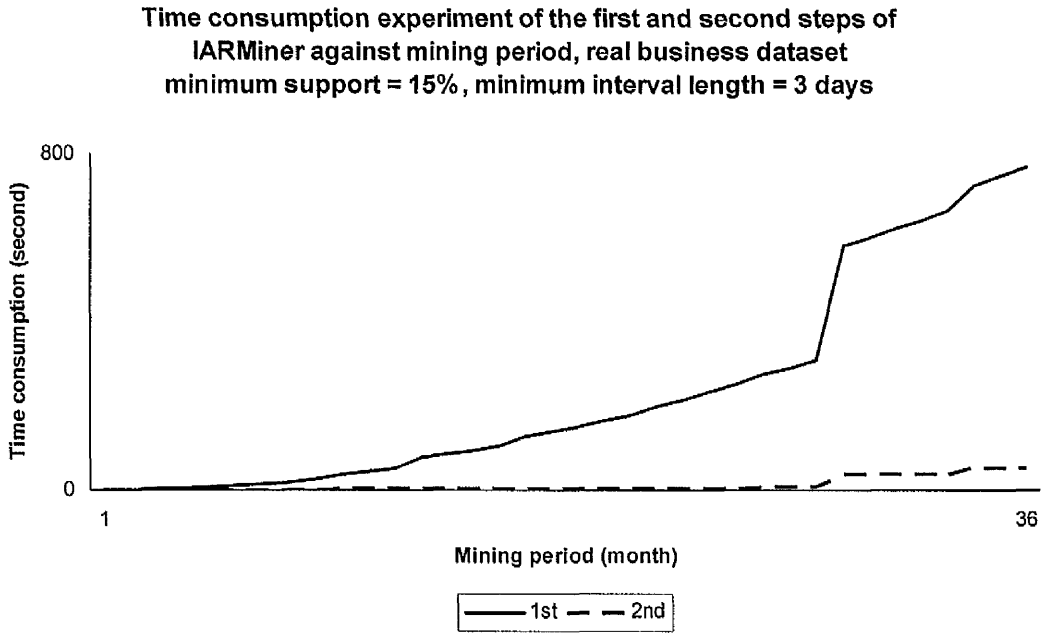


Figure 5.20 Time consumption of 1st and 2nd steps of IARMiner

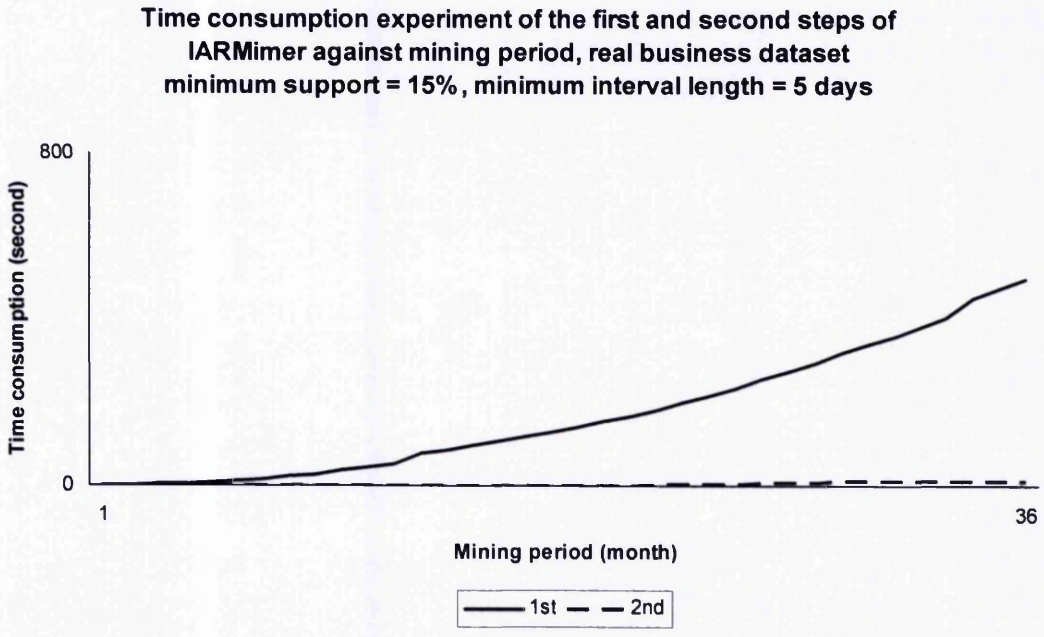


Figure 5.21 Time consumption of 1st and 2nd step of IARMiner

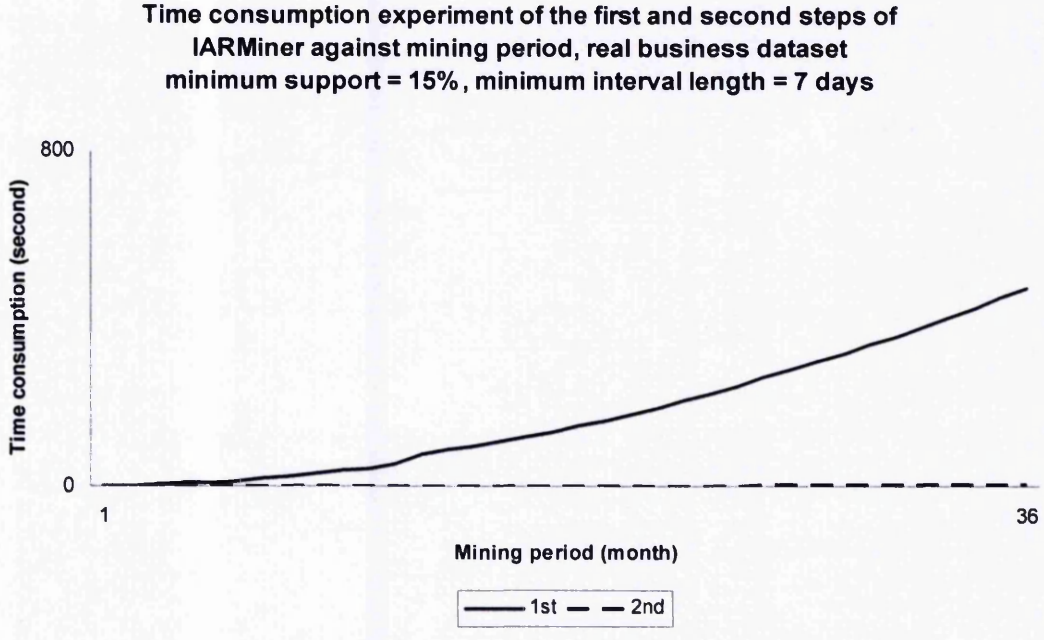


Figure 5.22 Time consumption of 1st and 2nd step of IARMiner

Figure 5.20, Figure 5.21 and Figure 5.22 are the results of experiments in which the minimum interval length equals three, five, and seven days. All these three charts point out that in comparison to the second step, the first step of IARMiner was really playing the leading role in the time consumption of the program. As mentioned before,

the first step of IARMiner is really a repeated regular association rule mining process running on every minimum length interval. The best way to reduce the time consumption of IARMiner is to choose a faster regular association rule mining algorithm (the algorithm Apriori [Agrawal and Srikant, 1994] was chosen in this research and experiments).

To evaluate the efficiency of IARMiner, a group of tests were set up to compare IARMiner with LISeeker [Chen, 1999] [Chen and Petrounias, 1998a]. As LISeeker discovers only the longest intervals of given association rules but IARMiner finds both association rules and their longest intervals together, Apriori [Agrawal and Srikant, 1994] was chosen to run firstly on each minimum interval length dataset to provide LISeeker the association rules.

After calculation, it was assured that the whole datasets to be treated could be loaded into main memory. This made Apriori run on the whole datasets without suffering disk I/O overhead, and had the efficiency of algorithm Partition [Savasere et al., 1995]. The response time was measured as the time which elapsed from the start of IARMiner loading the dataset to the end time of finishing the computation, and from the start of Apriori loading the dataset to the end time of LISeeker finishing the computation.

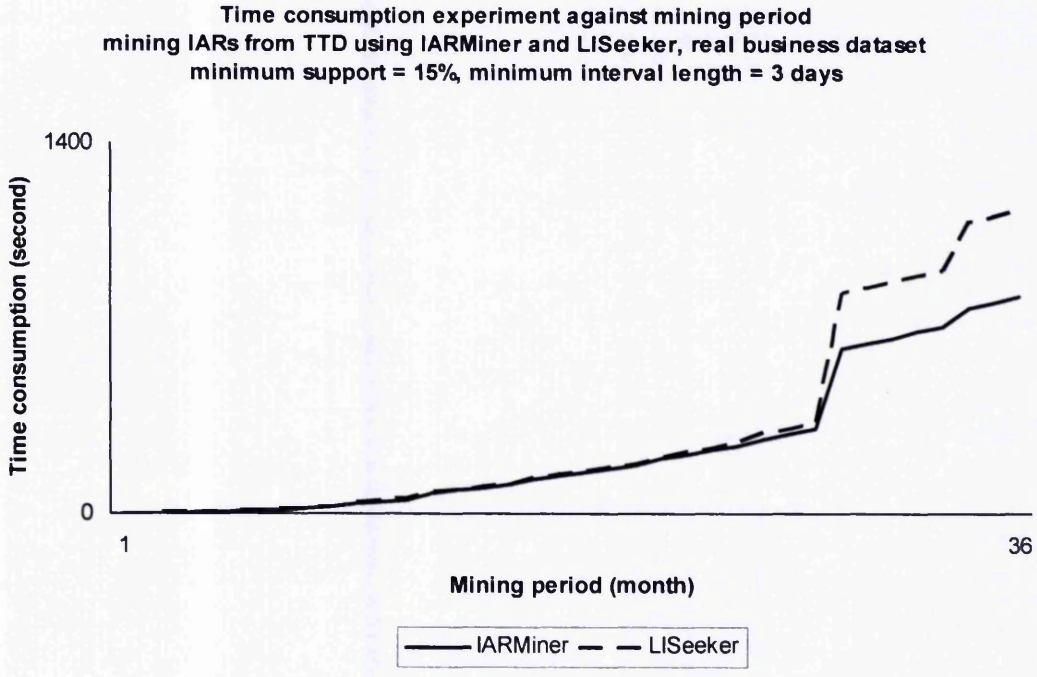


Figure 5.23 Time consumption of IARMiner and LISeeker

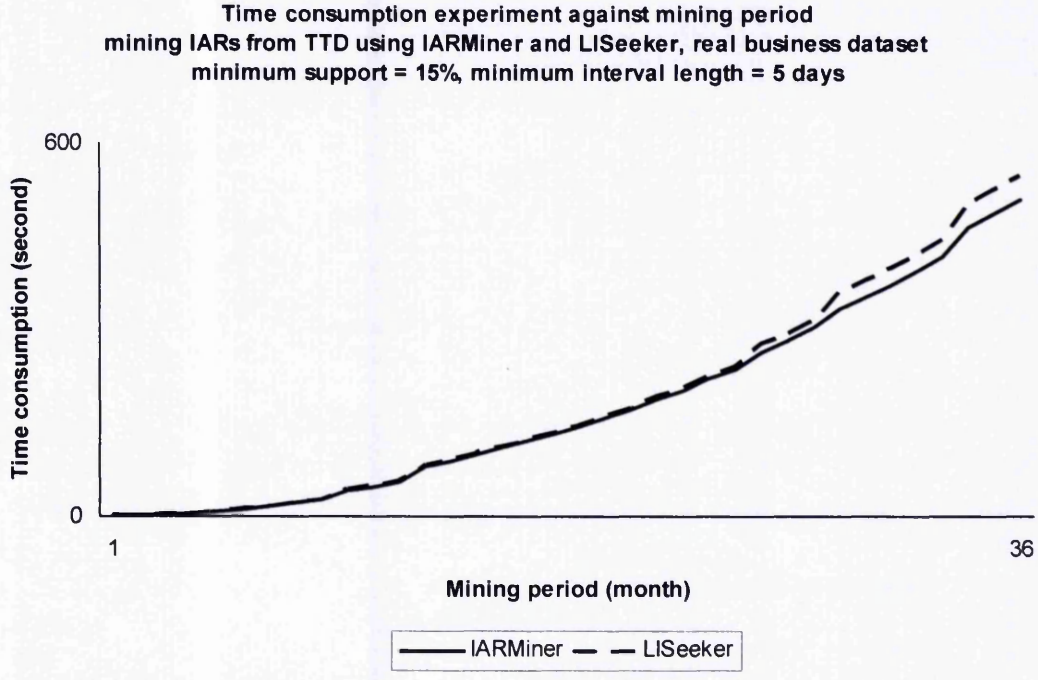


Figure 5.24 Time consumption of IARMiner and LISeeker

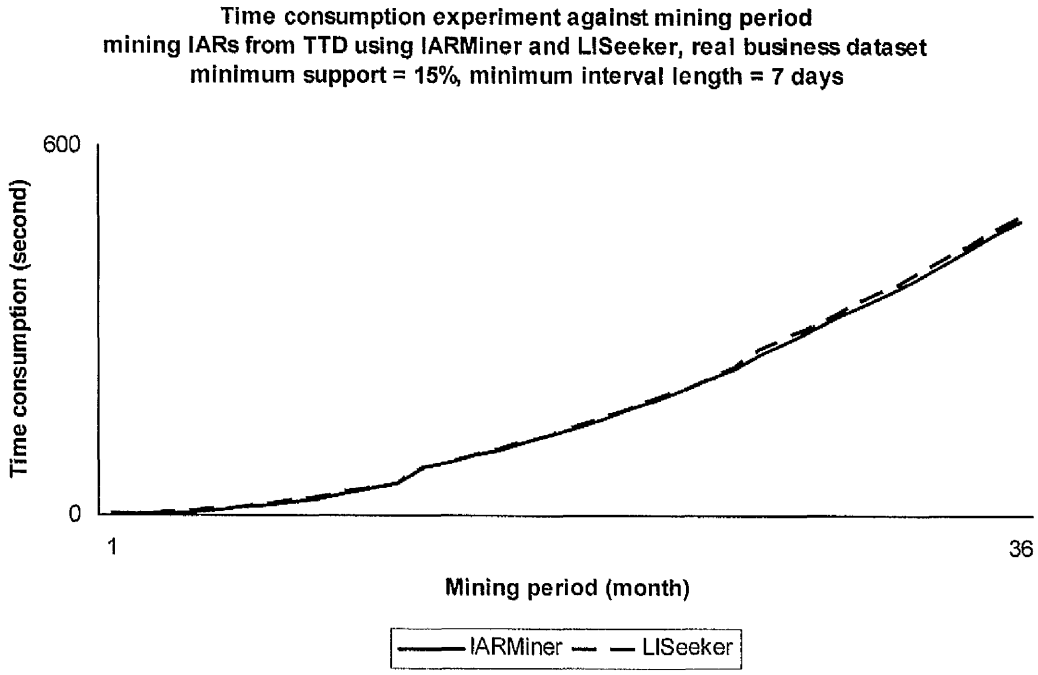


Figure 5.25 Time consumption of IARMiner and LISeeker

Figure 5.23, Figure 5.24 and Figure 5.25 are the experiment results, which compare IARMiner with LISeeker plus Apriori running on the real business dataset with the minimum interval length thresholds equal to three, five and seven days. It looks like IARMiner shows not much better performance than LISeeker plus Apriori. But by comparing these three figures, one can find that a shorter minimum interval length threshold provides better efficiency. This is due to the fact that with a shorter minimum interval length threshold, more interval association rules can be found; thus IARMiner's interval combination technique can show its better efficiency than LISeeker. As previous experiments already show, in comparison to the second step, the first step of IARMiner is the time-consuming, repeating the regular association rule mining process running on every minimum length interval, while the second step is really based on the distribution of interval association rules. Without finding interval association rules, both IARMiner and LISeeker plus Apriori are the process of Apriori running on every minimum interval length dataset. So, IARMiner shows better efficiency than LISeeker plus Apriori while there are interval association rules existing according to the thresholds; and the more interval association rules that exist, the better the IARMiner shows.

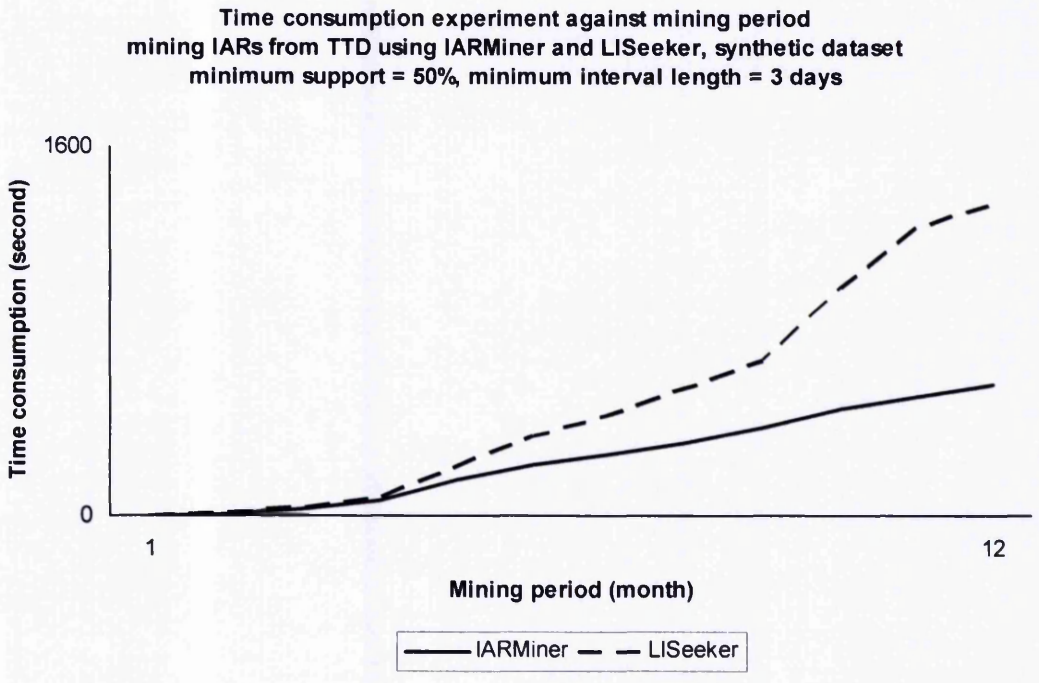


Figure 5.26 Time consumption of IARMiner and LISeeker

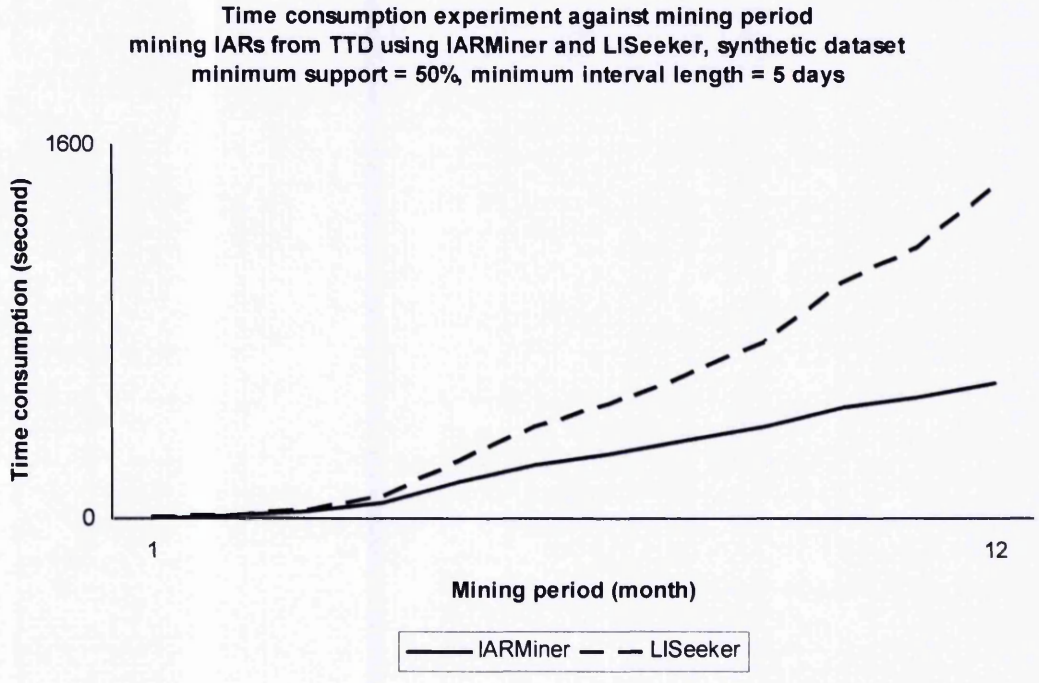


Figure 5.27 Time consumption of IARMiner and LISeeker

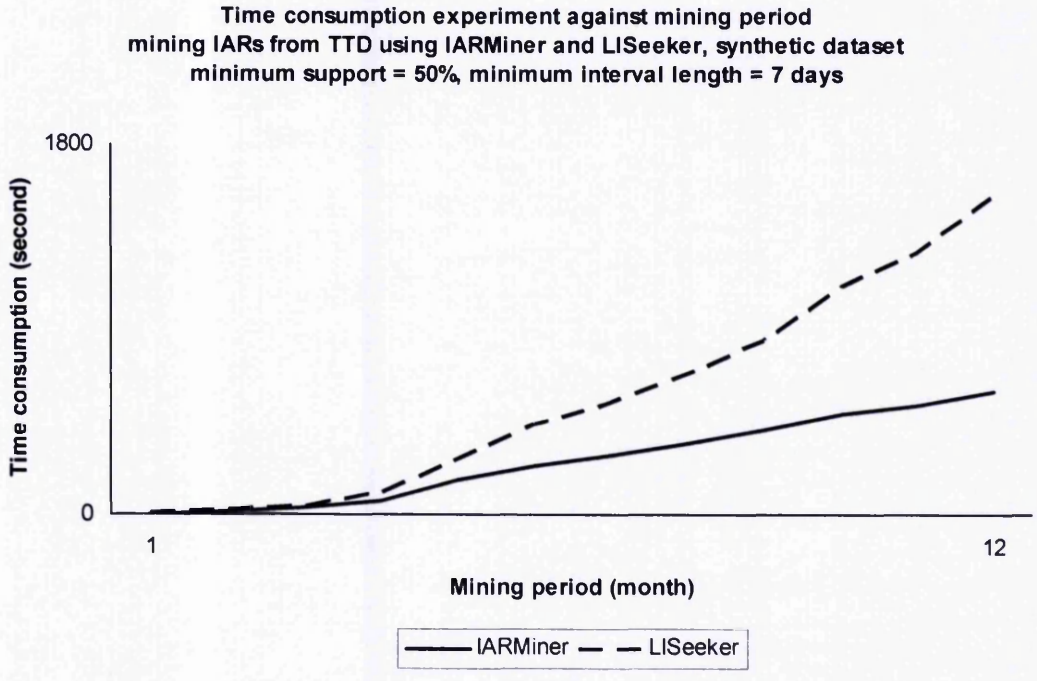


Figure 5.28 Time consumption of IARMiner and LISeeker

Figure 5.26, Figure 5.27 and Figure 5.28 are the same experiment running on the synthetic dataset. This time, IARMiner shows significantly better efficiency than LISeeker plus Apriori because more interval association rules were found in the synthetic dataset than in the real business dataset. It is again confirmed, that IARMiner is more efficient than LISeeker plus Apriori while interval association rules exist, and the more interval association rules that exist, the better the IARMiner shows.

5.7 Summary

In this chapter, the algorithm IARMiner was delivered through discussing the problem background, predicting the mining challenge, describing in detail, the algorithm implementation, and a range of evaluation processes. In the problem description, the problem of mining interval association rules was highlighted as an area of temporal association rule problems. Some previous research studies were reviewed and the special challenge of mining interval association rules was pointed out. Although the problem of mining interval association rules has a two-dimension solution space, one can not simply unite the association rule mining algorithm and the longest interval mining algorithm to try to find a solution.

Some related definitions for mining interval association rules, including interval association rule mining task, intervals, valid intervals, long interval, longest intervals etc., were then reviewed and redefined. The algorithm IARMiner was introduced from the longest interval searching technique and the association rule searching technique. A method called interval combination was introduced to IARMiner to fulfil the two-dimension solution space requirement.

The implementation part of this chapter introduced the programming method of IARMiner through its overall structure, step 1 structure and step 2 structure, in three parts. Finally, a real business temporal transaction dataset and a synthetic one were used to evaluate the algorithm IARMiner. Many experimental results were presented in clear charts with detailed explanations. The evaluation showed that the algorithm IARMiner is not only relatively affordable and efficient, but also has the potential to challenge the problem of mining both association rules and corresponding longest time intervals together, without any pre-given temporal or association rule clue. It delivered a reasonable response time, and also showed that its interval combination technique is more efficient than LISeeker. The algorithm IARMiner still has the potential to be improved by choosing a faster regular association rule mining algorithm for its first step.

6 A Tree-Projection Method for Mining Temporal Association Rules

6.1 Introduction

Since most level-wise association rule mining algorithms suffer the problem of scanning transaction datasets over multiple passes, as an extension of association rule mining, temporal association rule mining algorithms are facing the same difficulty. One of the most important methods for improving the efficiency of temporal association rule mining algorithms is to reduce the passes of scanning transaction datasets. A projected pattern base may be the solution. This chapter introduces a tree projection method called TI-tree to try to solve the problem.

The chapter discusses the problem firstly as the motivation for the work. Then, it points out the basic requirement of the projected pattern base to be used to solve the problem. Some different designs are then compared to choose the most suitable design, and in the discussion of its implementation, the chapter describes some key techniques of realising the design. Finally, the evaluation points out the effectiveness of the new method in comparison with other algorithms.

6.2 *Using the Temporal Itemset Base Instead of the Transaction Set*

6.2.1 Motivation

To get the most wanted association rules, a user may repeat the same mining process multiple times with different minimum support and minimum confidence thresholds. If a user needs 10 attempts at mining a transaction dataset using different thresholds to get the average $10-items$ association rules he/she wants, the dataset will have to be scanned $10\times 2^{10}=10{,}240$ times if he/she chooses an Apriori-like generate-and-test based association rule mining algorithm. If the transaction dataset has 100,000 transactions, the total transactions to be read and counted will be more than a billion. Unfortunately, many real business databases contain many more transactions and items. In addition to this try-to-get procedure, the same problem can happen to incremental datasets. Users always have to re-mine association rules with the same thresholds when new data comes to the database.

The problem also happens with temporal association rule mining applications, and may become even worse because the minimum interval length is added to the existing minimum support and minimum confidence thresholds. So, to discover the temporal association rules required by the user, the try-to-get procedure will have to be repeated more times than mining regular association rules. Scanning dataset with multiple passes is the heaviest burden of temporal association rule mining applications. Although modern computers become faster and faster, such a burden is still the reason for low efficiency.

6.2.2 Minimising System Consumption by Reducing Multi-Passing

Han et al. [Han et al., 2000] introduced a method called FP-growth, which compresses the database representing frequent itemsets into a tree-based structure, but retains the itemset association information, to transform the problem of scanning the transaction dataset to the traversing generated pattern base. This method greatly reduces the

expense of multi-scanning transaction datasets. Although it takes time to generate a pattern base (FP-tree), compared to the following mining process, the approach is still cost effective for regular association rule mining applications. But, since the FP-tree used by the FP-growth method is a compressed frequent itemsets base, which gives up some low count itemsets, it does not support the mining of incremental datasets because the low count itemsets can be frequent when new data come in.

The FP-growth method transforms the association rule mining procedure from the repetition of generate-and-test (which needs scanning transaction dataset multiple passes), to the traversing of a pattern tree. The same idea can also apply to temporal association rule mining techniques. To apply such a projected pattern base method to temporal association rule mining, the generated temporal pattern base also needs to contain temporal information.

6.3 TI-tree (Temporal Itemset tree)

The data structure to be introduced as the temporal pattern base, separates the whole temporal association rule mining process into two steps. As Figures 6.1 and 6.2 show, the introduced temporal pattern base introduces a middle part called the temporal itemset base.

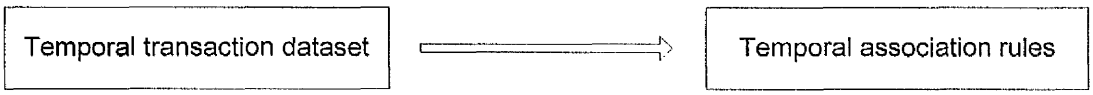


Figure 6.1 Temporal association rule mining process

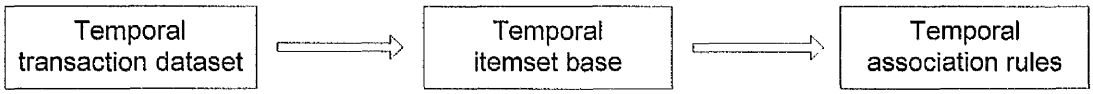


Figure 6.2 Temporal association rule mining process with temporal itemset base

The first step scans the temporal transaction dataset to generate the temporal itemset base, which contains all possible itemsets with the counts at every possible granular time piece. The second step takes the temporal itemset base as the input to search the association rules and their corresponding temporal features.

6.3.1 Design Requirements of the Temporal Itemset Base

For efficiently mining temporal association rules and reducing the costs of repeating the try-to-get procedure without multi-pass scanning of datasets, all itemsets and their counts with the corresponding temporal information from the source dataset, must be stored in the pattern base. Keeping all information without compressing those low count itemsets is not only done to maintain the ability to mine incremental datasets, but also to help the repeating try-to-get procedure of temporal data mining, because the user may reduce the minimum support or other threshold(s) during that procedure.

Based on the need to keep all itemsets and temporal information, three main requirements of designing the temporal itemset base arise, which are efficiency, capacity and size. The efficiency of the temporal itemset base must be high, because the retrieving speed of the itemset count and temporal information must beat the scanning of the original temporal transaction dataset. The capacity of the temporal itemset base must be big, since all itemset, count and temporal information is needed for the following mining processes. Finally, the size of the temporal itemset base must be acceptable and flexible. To keep all information well organised, the storage consumption (either in media or memory) of the base may be more than the original temporal transaction dataset, but the size must be acceptable or flexible to keep the temporal itemset base realisable.

In addition, there is some other information rather than itemset counts and corresponding temporal information, that must be kept in the temporal itemset base, this being, the amount of transactions of each granularity used to calculate the support and confidence, and all time granularity information throughout the whole time domain for mining temporal patterns.

6.3.2 TI-tree (Temporal Itemset tree)

To satisfy the above requirements, the suitable choice is a binary tree called a temporal itemset tree (TI-tree) here. A tree structure can satisfy the requirements and is more effective than the corresponding table-based or list-based structure because of

its unique searching efficiency. A tree structure can keep all count information from the transaction datasets well-organised and without duplication, which always happens in the list-based structure. Although the size of a tree structure is bigger than a corresponding list structure, it is still much smaller than a tabled one.

Because finding temporal association rules is the final purpose, the TI-tree structure has to be organised in time series. As Figure 6.3 shows, the itemsets with count information for a granular time piece, are projected as the nodes of a tree, and there are a number of trees in the time series.

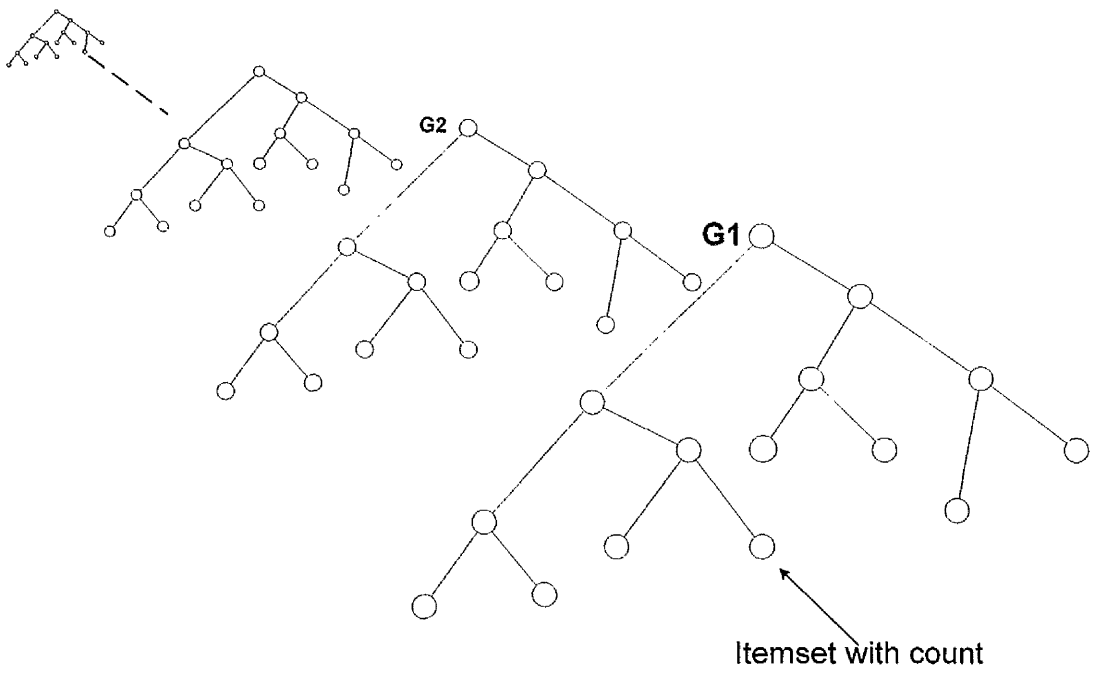


Figure 6.3 Tree organised in time series

There are two ways to organise granularity itemset trees, the first way being as shown in Figure 6.4, in which every granularity itemset tree contains the itemsets and their counts during the corresponding granular time piece, and all trees are linked together in the time series to form a bigger tree presenting the whole time domain. But, this approach has two drawbacks, the size of the structure occupies more storage space and it also makes it hard to search efficiently, since the same itemsets can repeat multiple times in different sub-trees.

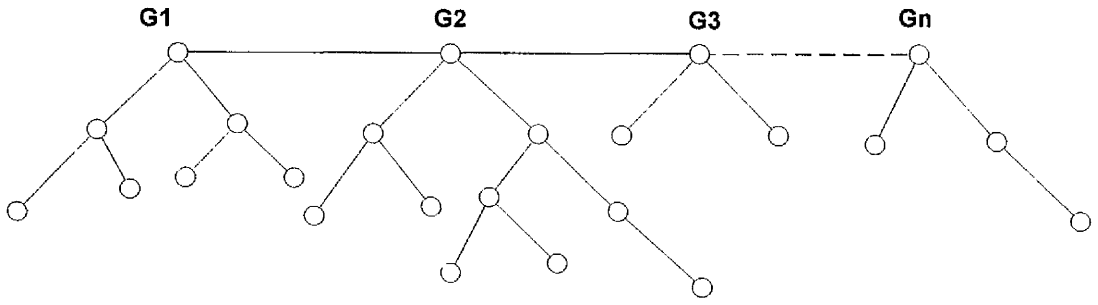


Figure 6.4 GI-tree linked in time series

Another approach is to store all granularity itemset trees (GI-tree) separately and introduce a list structure called G-list (granularity list) to keep all granularity information throughout the time domain. Each node of the G-list represents a granular time piece and points to a corresponding granularity itemset tree (Figure 6.5). This approach can improve the storage problems of Figure 6.4 by keeping granularity itemset trees separately and it also improves the searching efficiency.

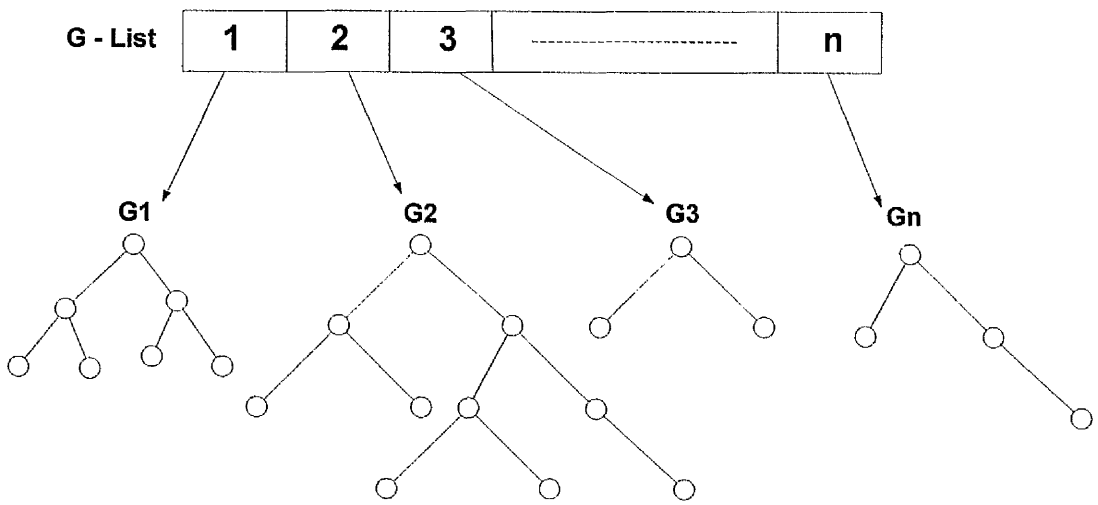


Figure 6.5 GI-tree linked by G-list

6.3.3 Mining Temporal Association Rules from the Generated TI-tree

Many existing temporal association rule mining algorithms can be reorganised to work with a TI-tree. A TI-tree provides them with improved efficiency for both mining incremental temporal transaction datasets and repeating the try-to-get procedure of discovering temporal association rules. To find the temporal patterns of

certain association rules using an algorithm like the longest interval mining algorithm LISeeker [Chen and Petrounias, 1999] [Chen and Petrounias, 2000a] with the TI-tree, the user can search each granularity itemset tree (GI-tree) in the sequence of the G-list throughout the time series domain, calculate the counts of the target itemsets stored in the GI-trees and get the longest intervals. To find the association rules for a given time duration, the user can perform a regular association rule mining process on the desired GI-tree(s), and the TI-tree also provides efficiency as it has already kept all counts information for every possible itemset. To search both association rules and the temporal information together using an algorithm like IARMiner with the TI-tree, the TI-tree provides unique efficiency for both the association rule and longest time interval mining processes, which will be discussed in detail later.

To efficiently discover association rules, Apriori [Agrawal and Srikant, 1994] uses a prune step to delete those candidate itemsets which cannot possibly be frequent in the candidate-generation function before scanning the transaction set. This method is easier to realise using the TI-tree to provide efficiency. As Apriori needs a prune step to delete those candidate itemsets whose sub-sets are not frequent during the last pass, the TI-tree finishes the task by avoiding those nodes whose parent nodes are not frequent. For example, if a GI-tree is generated like Figure 6.6, for mining certain frequent itemsets, the node BCD can be avoided if his parent BC is not frequent, and the nodes ABC, ABD and $ABCD$ can be avoided if the count of AB is less than the minimum support threshold. Using this method avoids traversing the whole tree while mining association rules and can save a great amount of precious time.

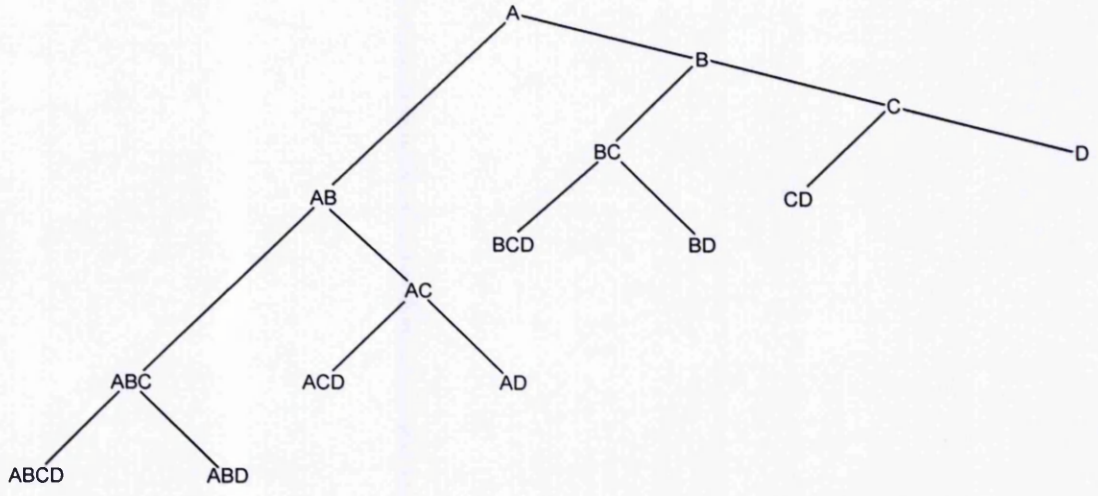


Figure 6.6 Example of GI-tree

To efficiently mine the longest intervals of a given association rule, the algorithm IARMiner can work with the TI-tree and shows better efficiency than LISeeker. The time consumption can be further reduced by searching only those intervals during which the current itemset node's ancestor nodes hold. The reasoning of such method is that if an itemset P_a does not hold on the duration T_a, any superset of P_a can not hold during P_a.

LISeeker [Chen and Petrounias, 1999b] makes the time complexity to $f = \mathrm{O}\left(k \cdot \frac{n^3 + 6n^2 + 11n + 6}{24}\right)$ while mining the longest intervals for a given association rule, where k is the size of the temporal transaction dataset and n is the amount of granularity time piece. The asymptotic time complexity of IARMiner is $f = \mathrm{O}\left(k \cdot \frac{n^2 + 3n + 2}{6}\right)$ while mining the longest intervals for a given association rule, where k is the size of the temporal transaction dataset and n is the amount of granularity time piece.

6.4 Improvements to the TI-tree

The TI-tree method discovers temporal association rules based on the time series counts information of every itemset, which is projected into the TI-tree from the source temporal transaction dataset. It shows better performance than multi-pass scanning of the temporal transaction set but also has limitations from two aspects, which are huge media and memory consumption, and low efficiency when a TI-tree becomes too big.

6.4.1 The Size Problem

Imagine an extreme situation of a temporal transaction dataset, where each granular time piece contains only one, but a big, transaction (itemset: $ABCD$). This will cause the corresponding GI-tree to generate 15 nodes (A, B, C, D, AB, AC, AD, BC, BD, CD, ABC, ABD, ACD, BCD and $ABCD$), whose storage space is much bigger than the original transaction dataset (Figure 6.7). If one transaction contains only one large size itemset consisting of n items, the corresponding GI-tree built from it will have $C_n^1 + C_n^2 + C_n^3 + \cdots + C_n^n = 2^n$ nodes. If a real business temporal transaction dataset has many such granularity time pieces, the storage and memory usage will explode.

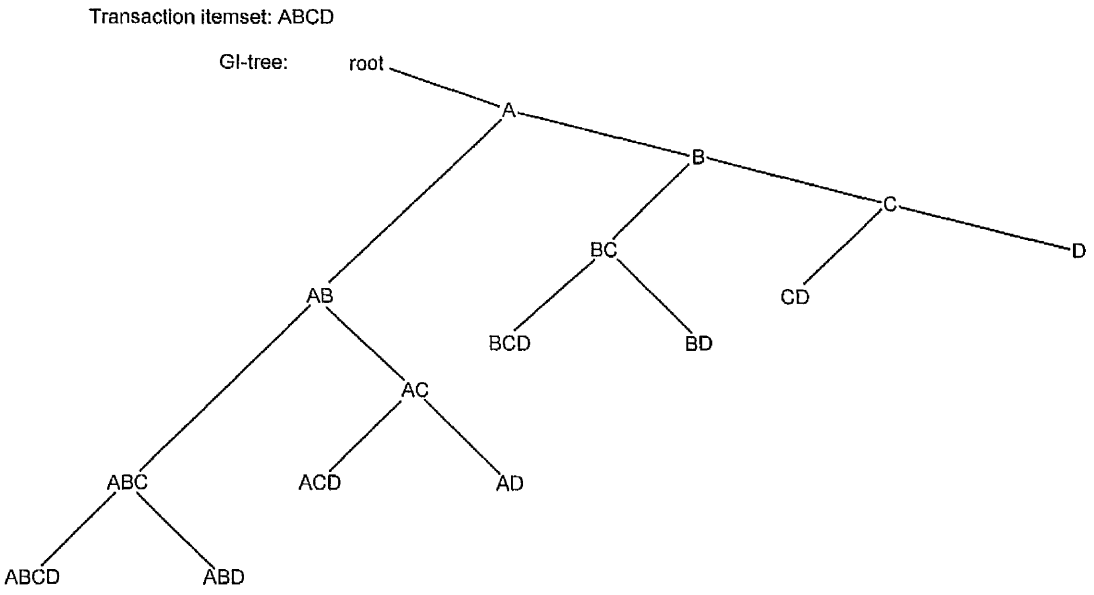


Figure 6.7 Example of GI-tree

6.4.2 The Complex Problem

There is the potential for a complex problem to emerge in real world applications when the user-specified granularity unit is relatively small. For example, a user may treat 'second' as the granularity time unit, but make the mining duration as days or months. In this situation, too many GI-trees will be generated and scanned, as Figure 6.8 shows. The mining process will keep tracking the reference processes between the G-list and GI-trees in Figure 6.8, and a large amount of system time will be wasted on traversing the TI-tree.

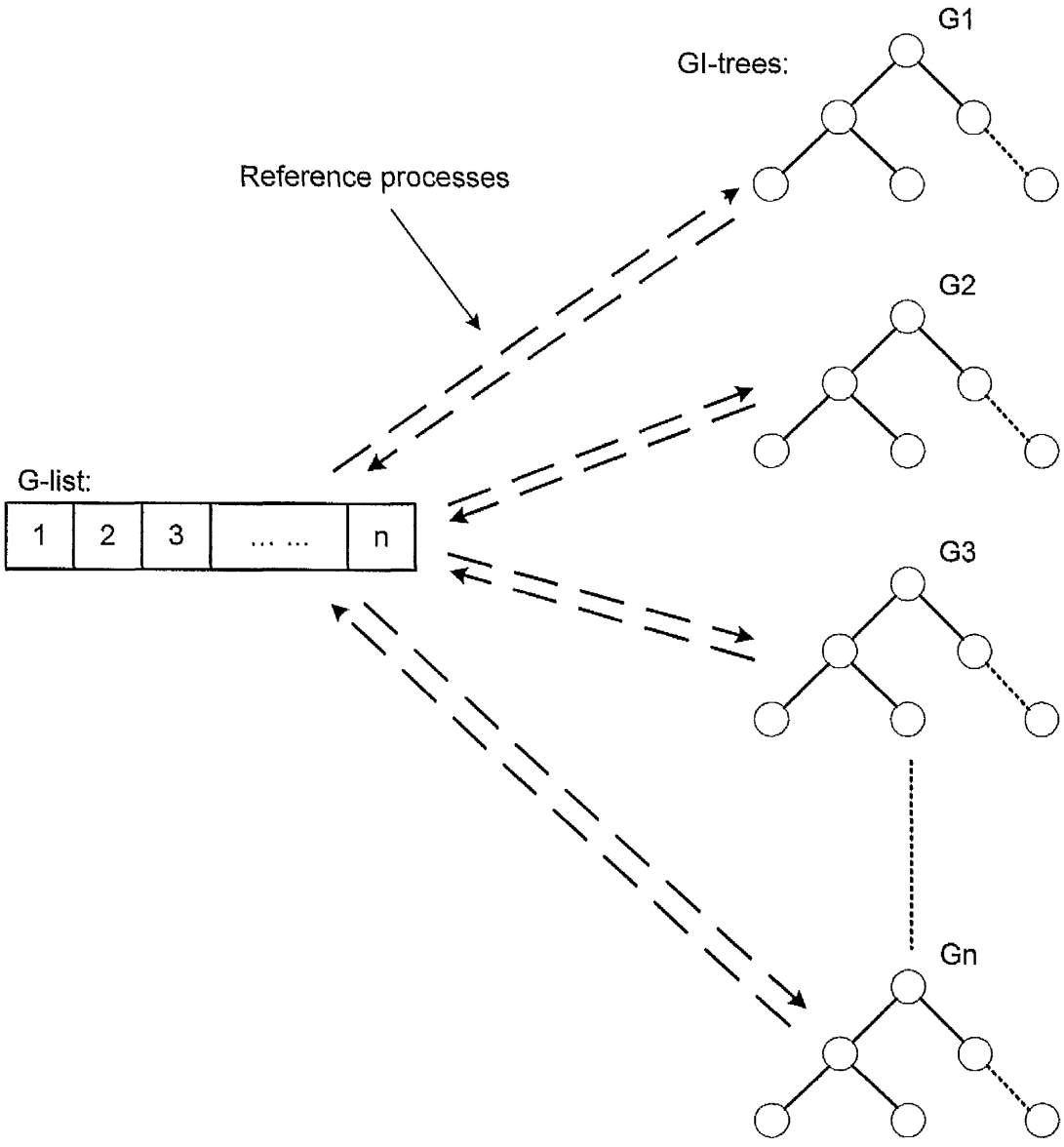


Figure 6.8 Complex problem of GI-tree and G-list

6.4.3 The Improved TI-tree

The above two problems may easily occur in real business applications. A superstore sales dataset can easily generate temporal transactions using low count but big size itemsets, and this circumstance makes the temporal association rule mining task work on long mining durations, but with a relatively short granularity dataset.

Such problems come from the 'one list linked multiple trees' structure of the TI-tree. The 'multiple trees' (GI-trees) waste the storage and memory spaces, and the 'one

list' (G-list) reduces the searching efficiency. So, the TI-tree method can be reorganised to a 'one tree linked multiple lists' structure to overcome the disadvantages.

Compared to the original TI-tree, the improved TI-tree has only one tree called I-tree (Itemset tree) but multiple lists called GC-lists (Granular count list) instead of multiple GI-trees with one G-list (Figure 6.9). The improved TI-tree records all itemsets information as the nodes of the I-tree, while the counts information of an itemset on each granularity is saved in the GC-list and pointed to by the corresponding tree nodes.

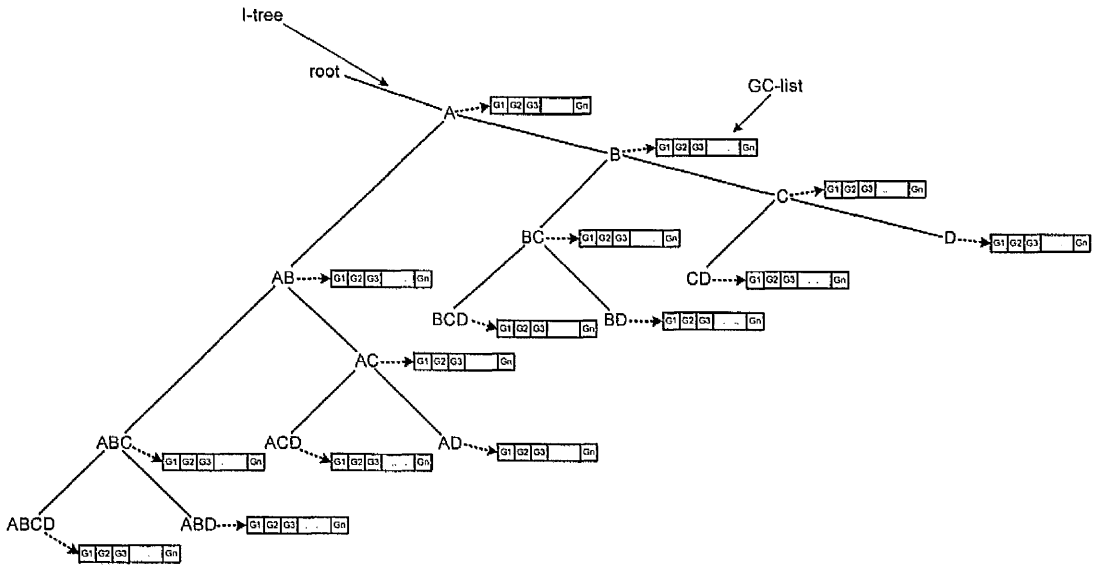


Figure 6.9 Organisation of GI-tree and G-list

In the improved TI-tree, not every granular count list (GC-list) has to be as long as the whole time series domain. A GC-list can avoid those list nodes whose corresponding granularities have no contribution to the count of the itemset. The improved TI-tree introduces another granularity transaction amount list (GTA-list) to keep the amount of transactions on each granularity throughout the time domain. Obviously, the GTA-list will have to be as long as the whole time domain. Such changing can save a great amount of storage space and improve the searching efficiency.

The improved TI-tree works as follows:

While mining association rules with temporal clues, go over every itemset node on the I-tree from the root only once, calculate the count information of each corresponding GC-list according to the thresholds, and finally generate the temporal association rules. (Those nodes of the I-tree whose ancestor is small to the minimum support threshold can be bypassed to create efficiency.)

While mining temporal patterns of a given association rule, search the corresponding itemset node of the I-tree, and calculate the counts on each granularity time piece stored in the GC-list according to the user-specified thresholds, to generate the wanted temporal patterns. (Currently, the most efficient method is the interval combining technique of IARMiner.)

While mining temporal association rules without any pre-given information, the situation is much like mining association rules with temporal information, the only difference being that without temporal information, the calculation has to be made all over the time series by using the interval combining method of IARMiner. (The method of bypassing those intervals, during which the ancestor itemsets are not frequent, can be used here to create more efficiency.)

6.4.4 The Potential Weakness of the TI-tree

The TI-tree reduces the expense of multiple scanning of transaction datasets for those unavoidable loop-repeating processes to provide high efficiency, but it still has some disadvantages.

Firstly, a TI-tree will grow bigger and bigger as more new items arrive. Although the growth is not significant while the user pays more attention to categories instead of certain item numbers, (which greatly reduces the amount of items,) a newly-introduced item still has the potential to double the amount of tree nodes by joining all other itemsets already in the tree. That is the worst situation and has hardly any chance of happening, but an over-large tree still needs to be separated into parts that can be stored and loaded into memory.

Another disadvantage of using the TI-tree happens when a mining process on an incremental transaction dataset has to change the granularity unit. While an existing TI-tree is generated from a 36-month temporal transaction dataset using the granularity unit of 'day', the user can increase the mining duration by projecting further transactions into the existing TI-tree without recounting previous transactions. But if the user wants a new mining job to be finished using the granularity unit of 'month', an additional process will be needed to combine the existing counts information from 'days' to 'months'. If the user wants the new granularity unit to be changed to 'hour', all previous temporal transactions will have to be recounted to generate a new TI-tree. So, it is important that the first TI-tree is generated according to a proper granularity unit. If the user can not decide which unit to use, choosing a shorter one will avoid rebuilding the TI-tree.

6.5 Implementation

The algorithm TI-tree is realised using the Java programming language on Microsoft Visual J++ 6.0 programming environment running on WindowsXP operating system.

The overall program structure, is shown in Figure 6.10. The program TI-tree generates the projected TI-tree including the I-tree and GTA-list using user-specified target temporal transaction datasets. After the generation of the TI-tree, the user can specify minimum support, minimum confidence, and minimum interval length etc., thresholds for mining temporal association rules from the projected TI-tree.

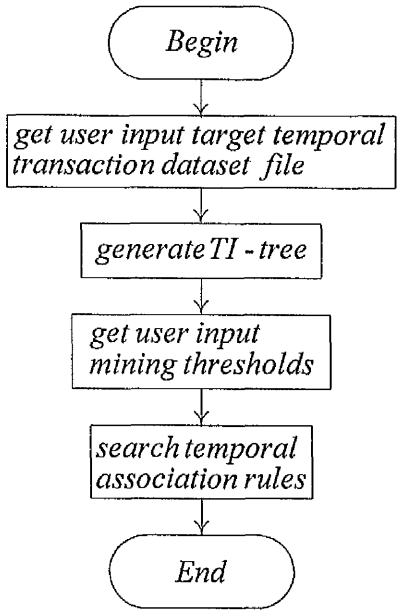


Figure 6.10 Overall program structure of TI-tree

6.5.1 Program Structure to Generate the TI-tree

Figure 6.11 depicts the program structure to generate the TI-tree, from which it can be seen that the program opens the target temporal transaction database file firstly, and then reads the data in each transaction, one by one. After reading a whole transaction, the program will increase the count information at the corresponding node of the GTA-list, which will record all transaction amount information in the granularity unit throughout the whole time domain. Then, the itemset in the transaction will be collected and sorted in ascending order. This sorting will help the tasks of both generating sub-itemsets and projecting into the tree. After the sorting, all sub-itemsets (sub-sets of the original itemset numbers) will be generated and linked in ascending order including the original itemset from the transaction. This itemset list will be inserted into the TI-tree one by one in ascending order. The process will repeat until the last transaction has been treated.

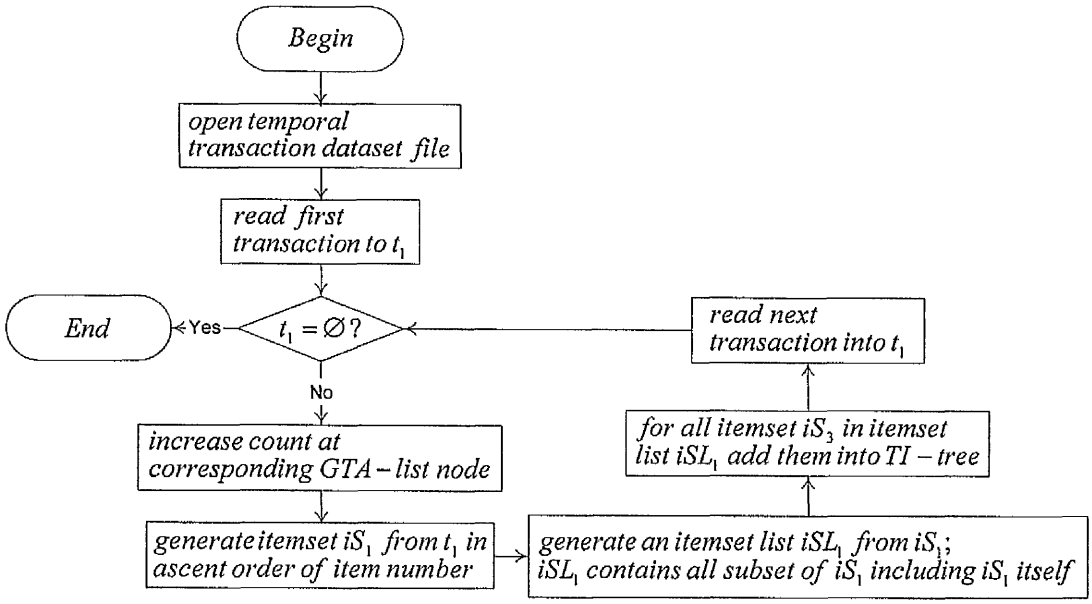


Figure 6.11 Program structure for generating TI-tree

Figure 6.12 is the process of generating sub-itemsets of a collected itemset. This process is much like the *join* operation of SQL (Structured Query Language). All sub-itemsets and the source itemset will be finally linked to an itemset list (called iSL_1 in Figure 6.12). The generation process starts from reading the first item number of the source itemset iN_1. The first item number will be enqueued to the itemset list iSL_1 as a 1-itemset, then the following item numbers from the source itemset will be united with those itemsets already in the list iSL_1 and enqueued to the list as new itemsets before enqueueing themselves to the list (as a 1-itemset). This process finishes when all item numbers from the source itemset are treated. The final itemset list iSL_1 will contain all sub-itemsets and original itemsets in ascending order.

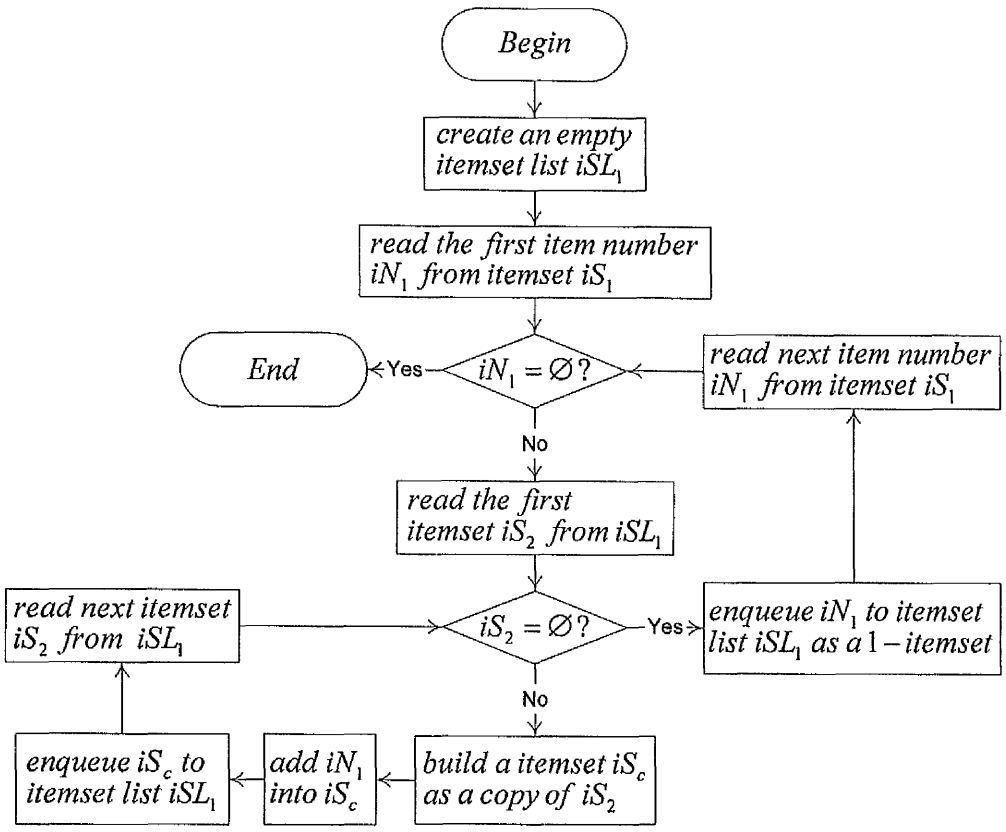


Figure 6.12 Program structure for generating sub-itemsets

After generating all sub-itemsets and building the itemset list in ascending order, all itemsets in the list will be inserted into the I-tree, thereby increasing the counts at the corresponding GC-list nodes. This process is depicted in Figure 6.13, and is seen to follow the procedure of finding a corresponding I-tree node and increasing the count in the associated GC-list, or building a new I-tree node with the GC-list and increasing the count. The program uses another procedure called 'compare' to determine the relationship between an itemset and a tree node. Depending on the relationships returned by the 'compare' procedure (which may be 'same', 'descendant of current node', 'elder brother of current node' or 'younger brother or descendant of younger brother of current node'), the program will increase the count in the GC-list, or traverse down to a descendant, or traverse to a younger brother, or build a new tree node with an associated GC-list. The program needs only four relationships because all itemsets within a list are linked and inserted into the tree in ascending order. The sorting processes reduce system consumption and program complexity.

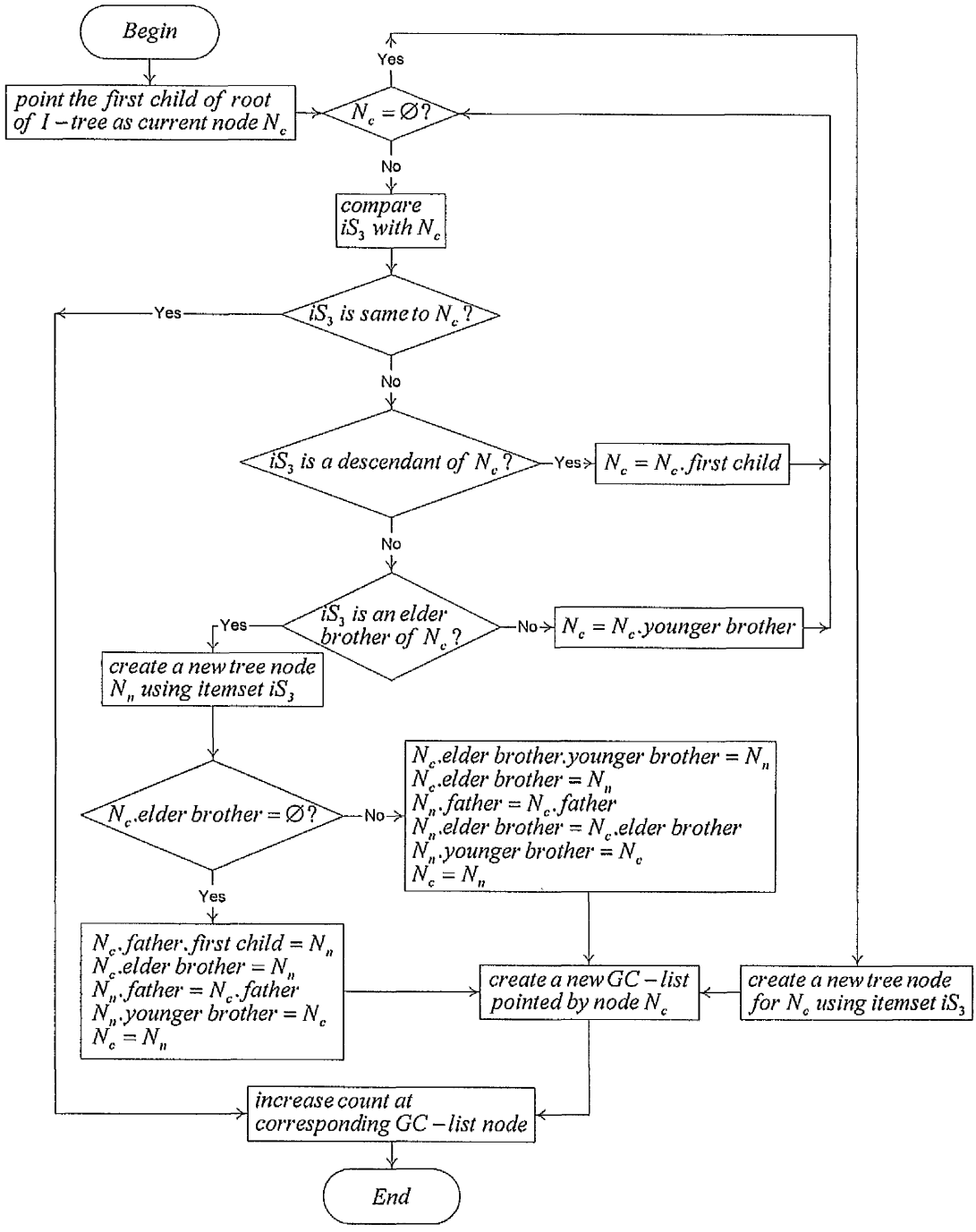


Figure 6.13 Program structure for inserting itemset into I-tree

Because the item numbers of all itemsets are also sorted in ascending order, the procedure of comparing an itemset with a tree node is relatively simple, as Figure 6.14 shows. Since the inserting procedure strictly moves down the tree branches from the root, the compare procedure needs only to compare the last item number of a node with the corresponding ordinal item number in the itemset being compared.

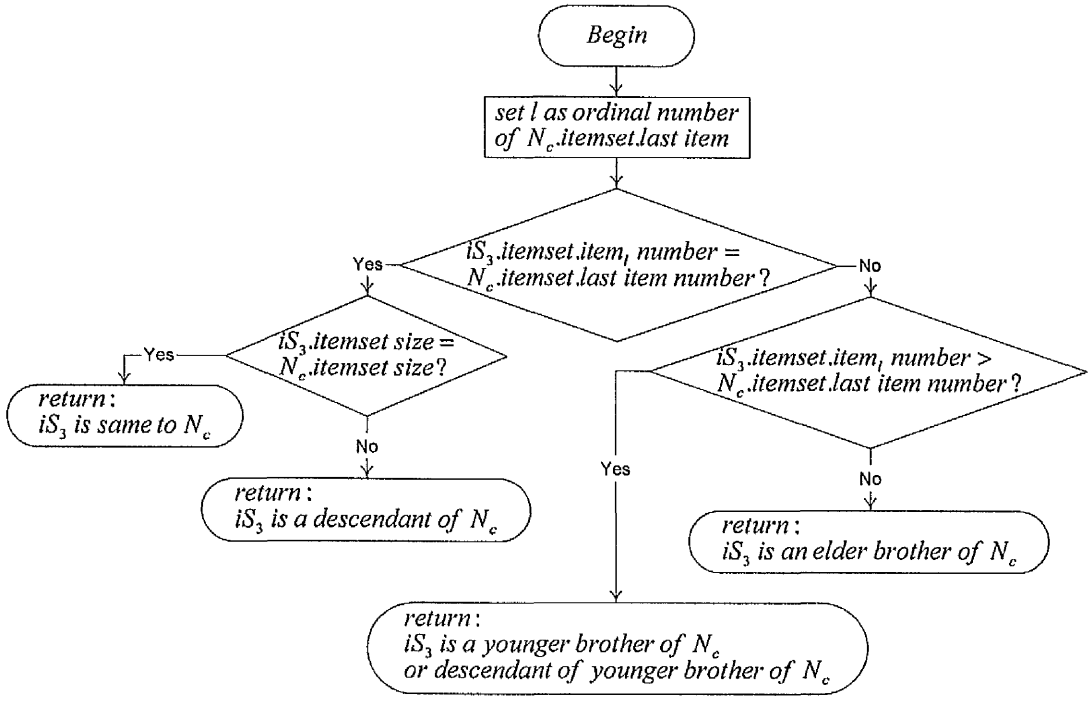


Figure 6.14 Program structure for comparing itemset with tree node

6.5.2 Program Structure of Mining Interval Association Rules from a Generated TI-tree

This section will explain the program structure for mining interval association rules from a generated TI-tree. As mentioned before, the mining of interval association rules from a TI-tree has two dimensions. The mining process traverses the I-tree to find frequent itemset(s), as well as searching for the longest interval(s) from a GC-list.

Figure 6.15 is the program structure for the beginning of mining interval association rules. When the I-tree is not empty, the mining process calls function *traverseTree*( ) to finish the job. In fact, because the longest intervals need to be mined at the same time as mining association rules, the program has to introduce the application of regressive invocation.

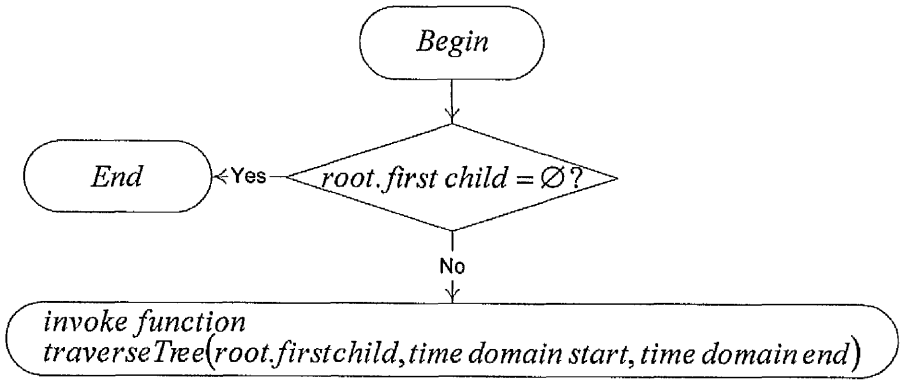


Figure 6.15 Program structure of beginning of mining

Figure 6.16 is the program structure of the regressive invocation. As the flow chart shows, the function *traverseTree*( ) continually invokes itself to go through each child and brother node to traverse the whole I-tree without repeating. Within the function, it first calls another function called *minelongestIntervals*( ) to find all possible longest intervals for the current node itemset. (The function *minelongestIntervals*( ) finishes the task as the algorithm IARMiner, which has been discussed in detail within the last chapter. The detailed flow chart of *minelongestIntervals*( ) can also be found in the last chapter.) Then, the program detects the existence of the first child of the current node, if it exists, and calls the function *traverseTree*( ) itself again for the first child. The same procedure applies to its younger brother. To achieve efficiency, the function *traverseTree*( ) is invoked with different parameters. As discussed, in the algorithm TI-tree, it is unnecessary to further traverse down to the tree branches when the current node itemset is not frequent during any possible intervals. Also, if the itemset of the current node is frequent during some intervals, it is unnecessary to search for the longest intervals of its descendants' nodes outside the intervals that the current node is holding. So, when the function *traverseTree*( ) invokes itself for a child or brother node, it transfers its holding intervals' boundary through parameters $start_1$ and end_1.

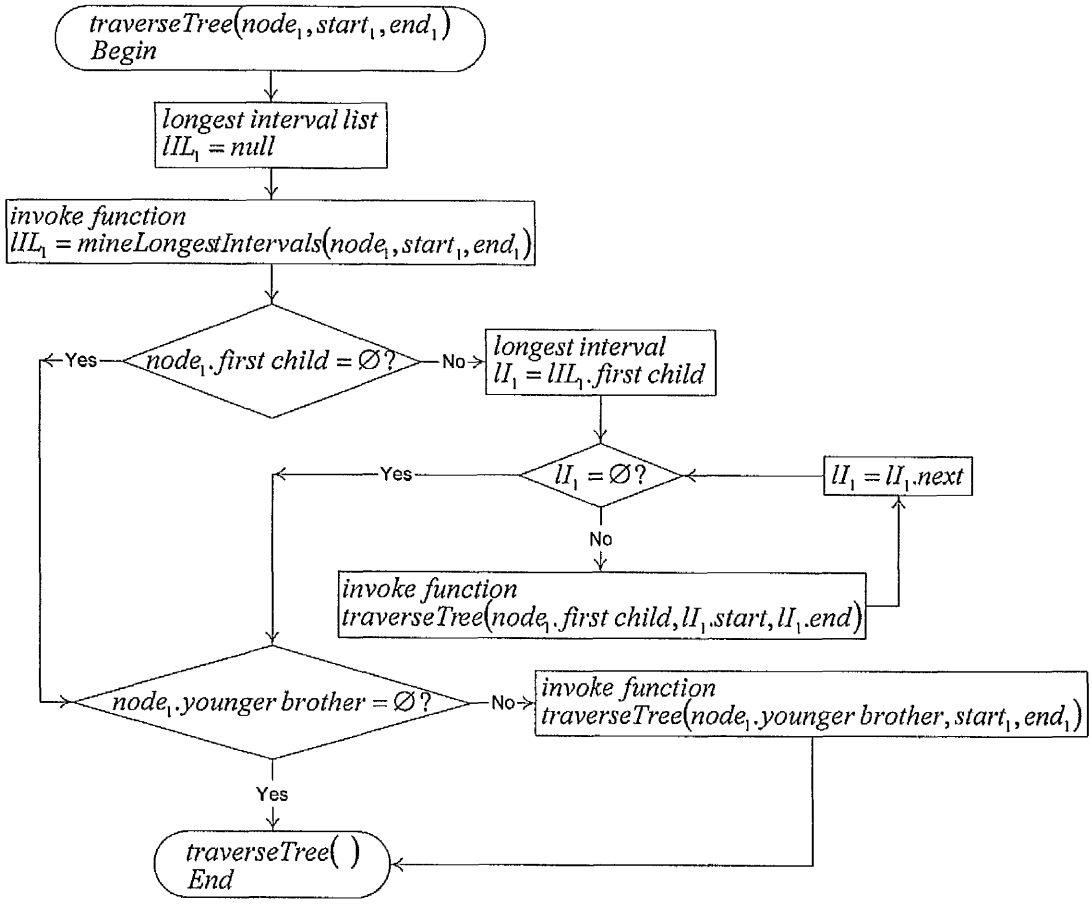


Figure 6.16 Program structure of regressive incovation

The detailed program code is included in appendix-b.

6.6 Evaluation

To evaluate the performance of the TI-tree, a set of experiments were conducted to quantify and measure the effectiveness of the actual algorithm. The algorithm TI-tree was implemented using the Java programming language on the platform of a laptop computer running Microsoft Windows XP professional operating system with Visual J++ 6.0 programming environment. The laptop was equipped with an Intel PentiumM 1.5GHz CPU, 60GB hard drive and 1GB memory.

6.6.1 Experiment Dataset

A synthetic dataset and the real business dataset, which is same as the one used for evaluating IARMiner, are used here to evaluate this tree-projection algorithm.

The real business dataset

The real business dataset came from a retail company, whose business was selling office copy machines, supporting parts and other office products. The transaction dataset covers the period from January 2003 to December 2005 - 36 months totally with 140,702 transactions containing 9,605 product IDs (item number). After categorising, there were 152 new categories (new item number) and 95,138 transactions remained for evaluation.

The synthetic dataset

The synthetic dataset used for the evaluation was designed to meet the requirements of a temporal transaction dataset, fulfilling the coverage of a one-year time period, setting the amount of items to 20 for fitting the generated TI-tree into main memory, and setting the most association frequent itemsets to reach 50% support. In the synthetic dataset, there were 14,637 transactions, with an average of 8 items in each transaction.

6.6.2 Time and Memory Consumption Experiments of Generating the TI-tree

As the TI-tree was introduced to speed up the temporal association rule mining process, the consumption time of building the TI-tree is important and must be evaluated firstly to obtain a reference point.

The response time was measured as the time that elapsed from the initiation of the execution to the end of finishing the process. Experiments were repeated 5 times to obtain stable values for each data point.

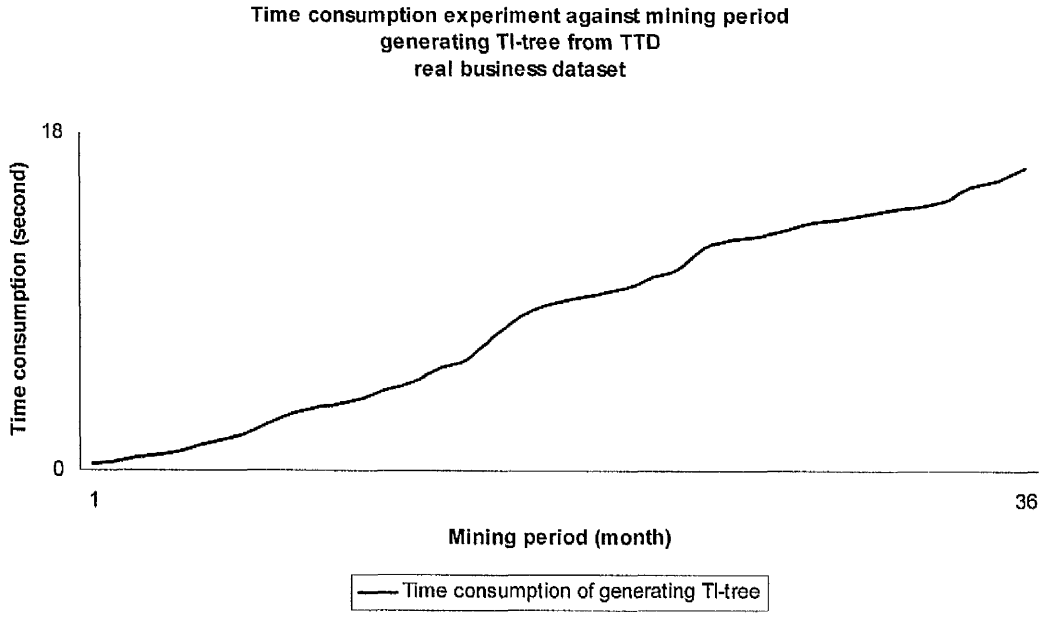


Figure 6.17 Time consumption of generating TI-tree

Figure 6.17 shows the time consumption of generating a TI-tree from different length temporal transaction datasets. The experiment was repeated using the temporal transaction datasets of different lengths that varied from 1 month to 36 months. The experiment result shows the time consumption increasing by a fixed amount of time as the length of the temporal transaction dataset increased by one each month. This is the most expected result as it shows the possibility of building the TI-tree for real world applications within acceptable time.

After the time consumption experiment, another key ability for realising the approach in the real world, that being the memory consumption, was tested. As Figure 6.18 shows, when generating the TI-tree for the first 24 months, there is a significant rise in memory consumption. But from the 25th month to the 36th month, the memory consumption remains almost constant, mainly because the initial 24 months' transactions bring in almost all items, and from the 25th month onwards, no more tree nodes are required for new items. The only memory consumption is related to the storage of new granularity itemset counts, which need much less memory space. This is quite an inspiring result, as it means that when using the TI-tree in real business applications, the demand on the memory will not impede the mining of incremental

temporal transaction datasets, as long as there is no significant introduction of new items.

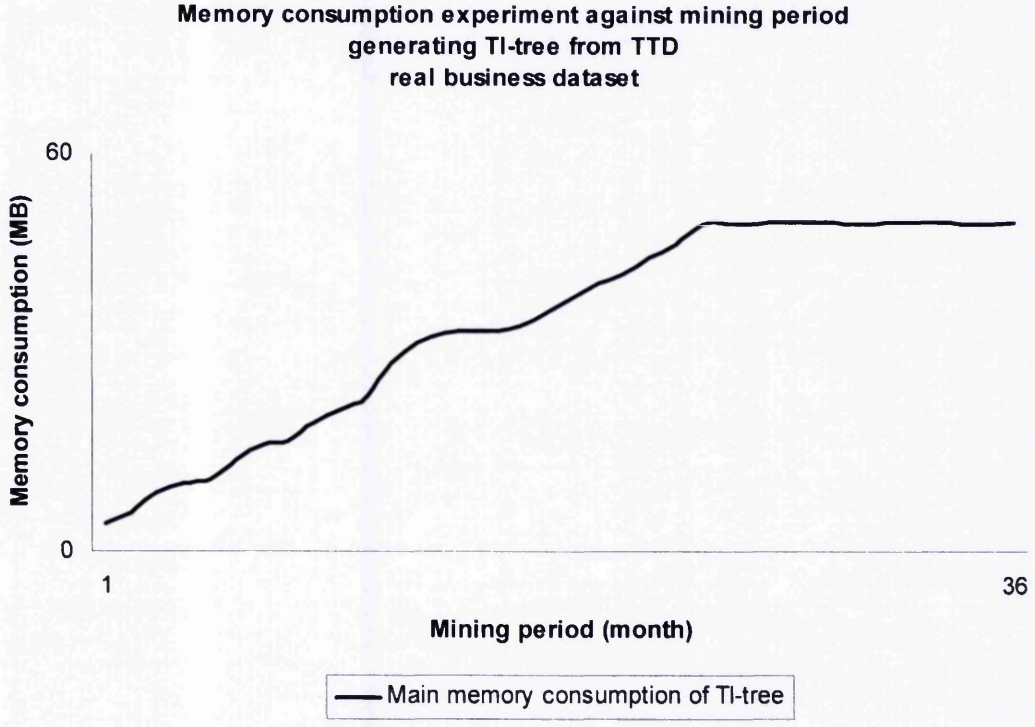


Figure 6.18 Memory consumption of generating TI-tree

6.6.3 Time Consumption Experiments of Mining Interval Association Rules from the TI-tree

Because there is no need to use system memory to build up data structures while mining interval association rules from a built TI-tree, only the experiments of time consumption are presented here.

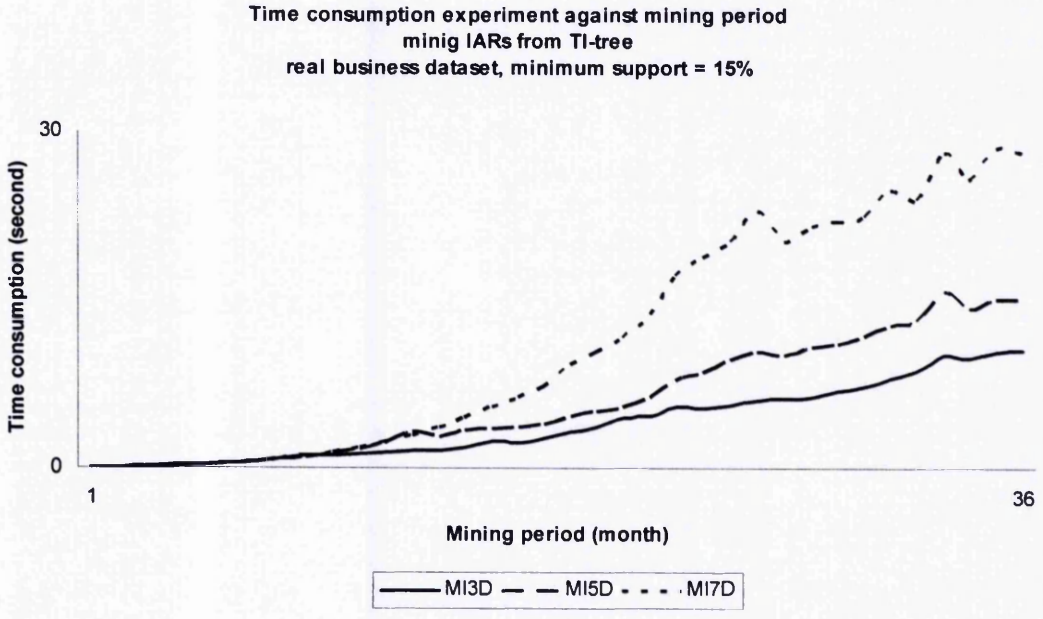


Figure 6.19 Time consumption of mining IARs from TI-tree

Figure 6.19 shows the result of the experiment of mining interval association rules from a generated 36-month TI-tree using the minimum support threshold at 15%, around which most interval association rules hold. As the mining duration increases, the mining process takes a reasonable response time. Comparing the different minimum interval length thresholds (which are represented in the chart as $MIxD$, x is the amount of granularities as the unit of 'day'), the longer the minimum interval length, the more the time consumed. This is because a longer minimum interval length always needs more calculation.

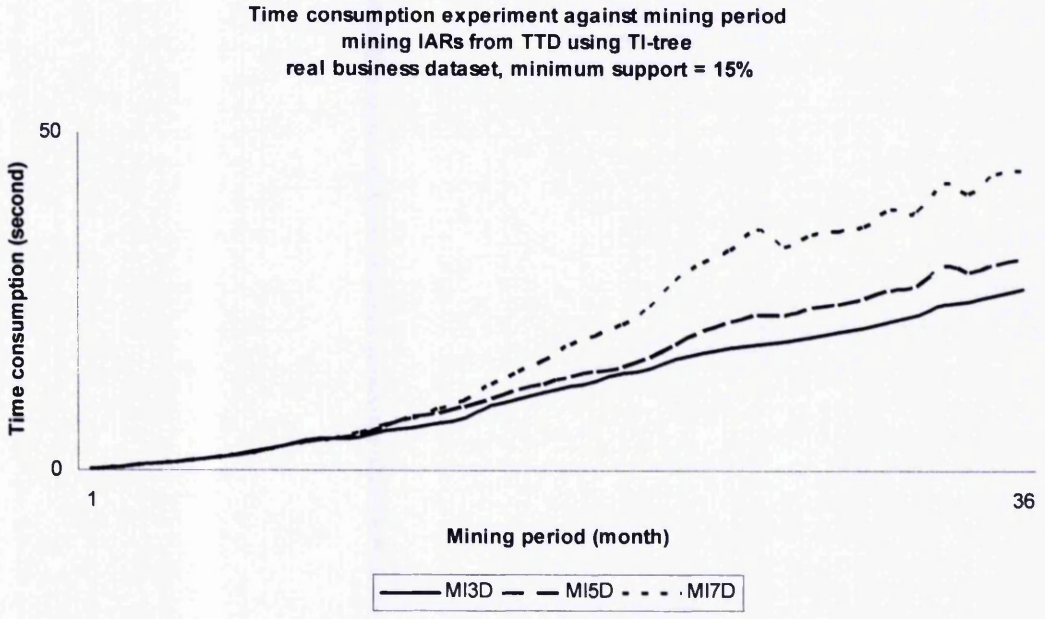


Figure 6.20 Time consumption of mining IARs from TTD using TI-tree

While Figure 6.19 shows the time consumption experiment result of mining interval association rules from a generated TI-tree, Figure 6.20 shows the total time consumption of mining interval association rules from the beginning of generating a TI-tree. The response time was measured as the time that elapsed from the initiation of generating a TI-tree to the end time when the last interval association rule had been found. Experiments were repeated 5 times to obtain stable values for each data point.

Because the time consumption is extremely low, the time consumption experiments were repeated using both TI-tree and IARMiner algorithms, and the results were compared to gain a clear view.

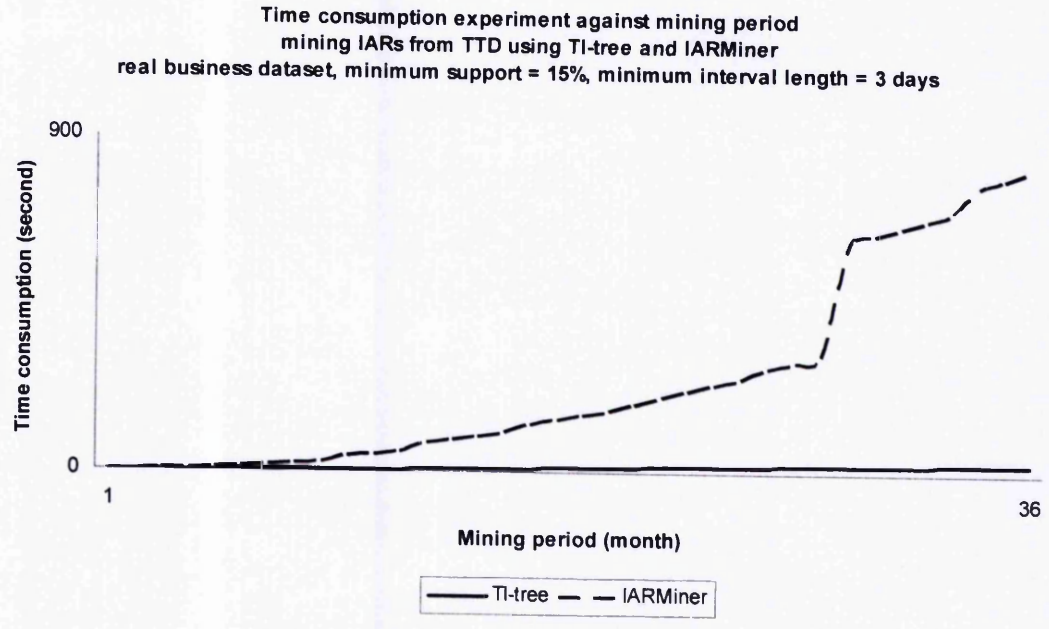


Figure 6.21 Time consumption of TI-tree and IARMiner

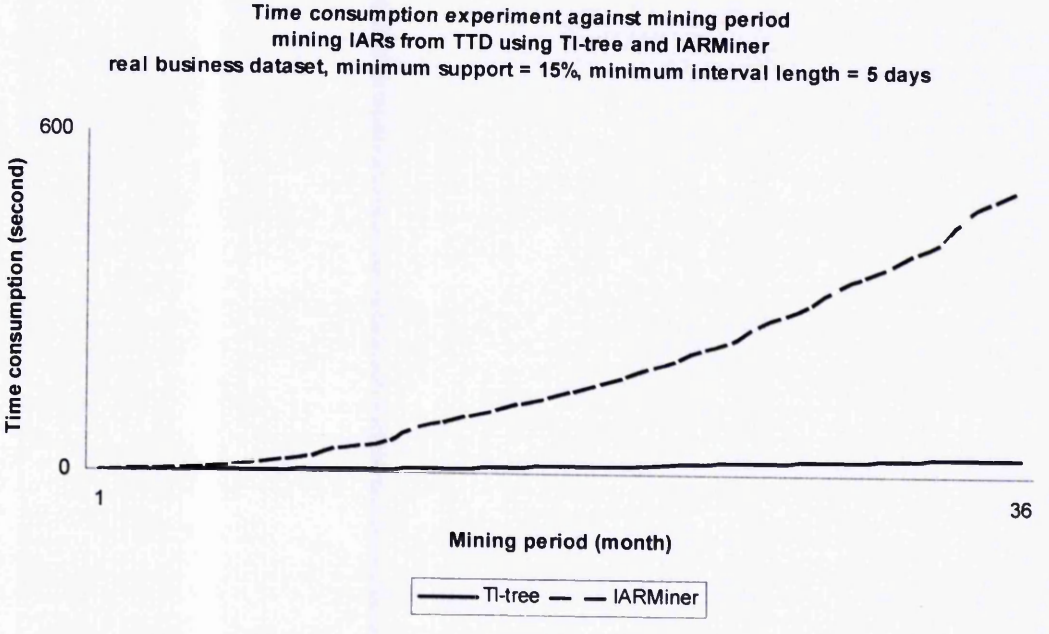


Figure 6.22 Time consumption of TI-tree and IARMiner

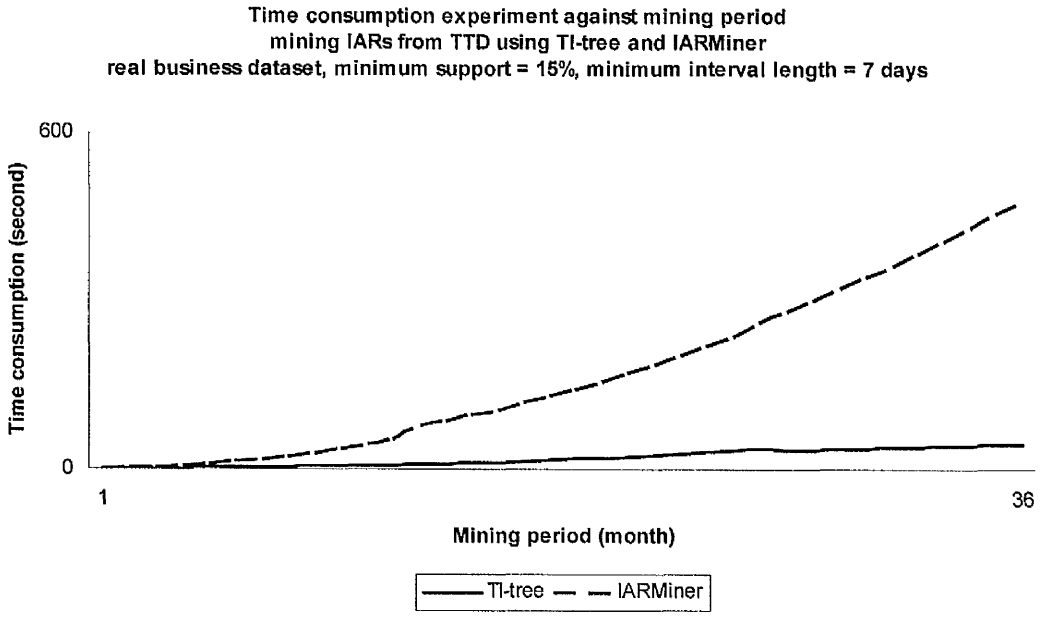


Figure 6.23 Time consumption of TI-tree and IARMiner

Figures 6.21, 6.22 and 6.23 are the experiment results repeated on the same temporal transaction dataset with different minimum interval length thresholds. All of the three figures show that the TI-tree algorithm provides much better efficiency while mining interval association rules, and the longer the mining period, the better the result. This is mainly because of the TI-tree's triple acceleration methods. Firstly, using the temporal itemset tree instead of a transaction dataset reduces the time consumption of multi-scanning the transaction dataset. Secondly, using the method of bypassing certain tree nodes whose parent node is not frequent, reduces the time consumption further, by avoiding traversing every tree node. Finally, the method of bypassing certain time intervals which cannot possibly be long, delivers more efficiency.

The experiments on the real business dataset were not finished with longer and larger transaction datasets because the TI-tree should be fixed in the system's main memory to ensure accurate readings. The following experiment was performed on the synthetic dataset, which set the amount of items to 20 for fitting the generated TI-tree into main memory.

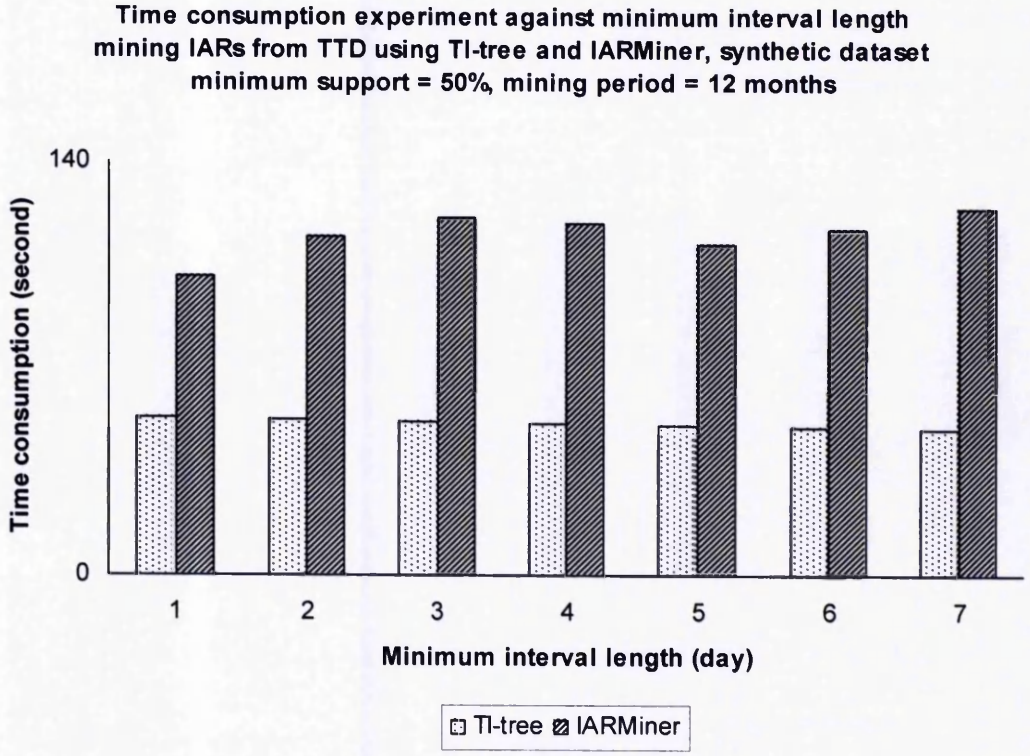


Figure 6.24 Time consumption of TI-tree and IARMiner

As Figure 6.24 shows, both TI-tree and IARMiner were performed on the synthetic dataset to provide the comparison. The algorithms treaded the synthetic dataset spreading 12 months period with 50% minimum support and one to seven days of minimum interval length thresholds. As it shows, with all minimum interval length thresholds, TI-tree provides better efficiency than IARMiner. TI-tree took only about half the running time of IARMiner.

6.7 Summary

This chapter introduced a method called TI-tree to try to make temporal association rule mining more efficient. The TI-tree uses a projected tree to record all itemsets and their counts information at each time granular piece throughout the time domain. This tree projection method transforms the multi-pass scanning of original temporal transaction datasets to the traversing of the generated temporal itemset base to avoid the expensive job of multi-pass scanning of the transaction dataset in temporal association rule mining.

The chapter first reviewed the problem of current temporal association rule mining algorithms. Then, the idea of using the temporal itemset base was introduced to solve the problem. Several design possibilities were discussed, the best one being chosen according to the requirements of the temporal itemset base, which are the efficiency, the capacity, and the size. Furthermore, based on the three design requirements, some important changes have been made to the designed TI-tree. The improved TI-tree provides more efficiency. A set of program structure diagrams were presented as they represent the key stages in realising the TI-tree. Finally, a set of experiments were performed using both real business and synthetic temporal transaction datasets. The evaluation shows that the results are inspiring although some disadvantages of the TI-tree still need more research in order to be overcome.

7 Conclusion and Future Work

7.1 Introduction

The research work undertaken in this thesis makes two contributions to current data mining research. The first is the interval association rule mining algorithm IARMiner, which is the first algorithm to discover certain kinds of temporal association rules without a pre-given association rule or temporal feature clue. The second contribution is the tree projection method TI-tree, which introduces the method of projecting the temporal transaction dataset into an organised temporal itemset tree to accelerate temporal association rule mining tasks. To achieve these research efforts, related background including data mining, association rules, association rule mining, temporal data mining and temporal association rule mining were reviewed. A set of experiments were performed using both synthetic and real business datasets to evaluate IARMiner and TI-tree. Besides the encouraging results, the evaluations also showed up areas for future research.

7.2 Summary of Background Research

The literature review was comprised of a wide search of the background and conceptual requirements for this thesis.

Firstly, Chapter Two illustrated data mining technology in several different aspects: data mining motivation, data mining and knowledge discovery in databases definition, data mining approaches, the data mining process, and data mining techniques. In this way, a complete and comprehensive understanding of the data mining area was provided to support the major concern of this thesis.

Secondly, as one of the most important mining techniques of data mining and the concern of the research area of this thesis, the concept of association rules, and the techniques for association rule mining, were presented and discussed in Chapter Three. As the core problem of association rule mining, the issue of frequent itemsets was considered. Then several algorithms, including AIS, SETM, Apriori, AprioriTID, AprioriHybrid, Partition and Sampling were explored in detail since they differ from each other, mainly in the way they search for frequent itemsets. A comparison of these algorithms was also provided. As most Apriori-based algorithms use the process of 'generating and counting candidate itemsets', an alternative way of mining frequent itemsets without candidate generation and mining association rules using logical operations were introduced at the end of Chapter Three.

Finally, as the other concern of this thesis, temporal data mining forms the main content of Chapter Four and appears to support the research area of temporal association rule mining. To establish the background, an overview of temporal data mining was given, followed by a discussion of the motivation.

Chapter Four studies temporal data mining in four classified groups: association rules, classification, clustering, and prediction. Two related issues - temporal features and representation of temporal features - were explored in detail in order to support the

research of this thesis, and the literature relating to temporal association rule mining techniques was also discussed.

7.3 Summary of the Research Work in This Thesis

7.3.1 Discovering Interval Association Rules

The first contribution of this thesis is the interval association rule mining algorithm IARMiner presented in Chapter Five. As an extension to association rules, the inclusion of the temporal dimension forms the area of temporal association rules. Previous research classifies the temporal features relating to association rules into two forms: interval and periodic. It also classified the mining tasks of temporal association rules into three groups, which are: discovering association rules for a given temporal feature, searching the temporal features of a given association rule, and mining temporal association rules in certain forms (e.g. interval association rules and periodic association rules) without any pre-given information. Some research work has already focused on the first two groups of mining task. But since it is believed that it is too expensive to directly search temporal association rules without pre-given temporal features or association rules, the third group of mining tasks that concern temporal association rules of certain kinds, such as interval association rules, were not touched. Relating to the temporal features of association rules, this group of mining tasks include the mining of interval association rules and mining periodic association rules. IARMiner is used for the first, mining interval association rules from temporal transaction datasets.

The mining task can be defined as "Given a set of time-stamped transactions (D) over a time domain (T), minimum support (min_sup), minimum confidence (min_con), and minimum interval length (min_len), the problem of mining interval association rules ($IARs$) is to find all possible association rules (ARs) and all their corresponding longest time intervals during which the association rules hold". In relation to the two dimension solution space which is mining association rules and finding their longest intervals, a method called interval combination was developed in this research to effectively search all the longest intervals of those association rules

found during the minimum interval length. In other words, it can be seen as "finding all minimum length interval association rules, then the minimum interval association rules growing to the longest interval association rules". This involves the definition of combinable intervals, which is used by IARMiner to generate interval association rules from minimum length interval association rules. IARMiner uses such an interval growth method to solve the problem of mining both association rules and their longest interval together.

The evaluation showed that the algorithm IARMiner is not only relatively affordable and efficient but also a try to challenge the problem of mining both association rules and corresponding longest time intervals together, without any pre-given temporal or association rule clue. It delivered a reasonable response time that is better than similar works, and also a possible way to solve the real business interval association rule mining problems, and still has the potential to be improved by choosing a faster regular association rule algorithm.

7.3.2 A Tree Projection Method for Mining Temporal Association Rules

Another contribution of this thesis is a tree projection method called TI-tree that helps to mine temporal association rules efficiently. Since most temporal association rule mining algorithms use the process of 'candidate generate and test', the problem of scanning transaction datasets using multiple passes is unacceptable. This problem greatly reduces the efficiency of temporal association rule mining algorithms and limits the applications in the real world. To solve the problem, a projected patterns base seems to be the solution. This projects the transactions from a source dataset into an organised pattern base, where the associations between the items are presented as the patterns with the counts. Thus, the temporal association rule mining algorithms does not need to multi-scan the transaction dataset to get count information about a pattern. This method transforms the problem of multi-passes scanning transaction datasets to the traversing of the pattern base, and can make for efficiency.

The TI-tree is a possible pattern base solution for temporal association rule mining. It was designed to fulfil the special requirements for use with temporal association rules. It keeps itemset information with count information in time series, so that all information needed by mining temporal association rules is well-organised and presented in the projected pattern base to avoid the multi-scanning of the original transaction datasets. The reason for using a tree-based temporal pattern base rather than others is the searching efficiency of a tree structure. Traversing a tree is much less costly than multi-scanning transaction datasets. The TI-tree keeps count information in the duration of each granular time piece, which is the smallest time unit for a certain temporal association rule mining task. This also provides the TI-tree with the ability to mine incremental temporal transaction datasets.

A set of experiments was performed using both real business and synthetic temporal transaction datasets. In comparison with the IARMiner, the TI-tree is far more efficient, and demonstrates much potential application in real business.

7.4 Future Research Areas

7.4.1 Mining Periodic Association Rules

Since IARMiner is the algorithm for mining interval association rules from temporal transaction datasets, the other form of temporal feature - periodic - remains unsolved. In this case, the interesting temporal feature is a set of regular intervals in cycles, during each of which the associations exists [Chen and Petrounias, 1999]. For example, a periodic association rule could be 'in the last month of every year, most vouchers and gift cards can be sold together'.

Chen and Petrounias [Chen and Petrounias, 1999] introduce the periodic time composed by three essential features which are cyclicity, granularity and interval range. So, they define a periodic time as valid with respect to an association rule if there are no less than the specified minimum frequency of intervals in the time domain which are strictly long with respect to the association rule. The introduction of new threshold minimum frequency brings new challenges. To effectively mine

periodic association rules, a new method like the interval combination of IARMiner may be necessary.

7.4.2 Developing the Memory Management Mechanism for the TI-tree

Although the TI-tree shows good efficiency for temporal association rule mining, the unavoidable disadvantage of the TI-tree is the size. As mentioned in Chapter Six, any newly arrived item has the potential to double the nodes in the TI-tree. Although the chance of that is relatively small since the user pays more attention to categories instead of certain item numbers (which greatly reduces the amount of items), an effective mechanism to partially load a TI-tree and exchange the parts is necessary, especially for mining large temporal transaction datasets.

There are two possible methods to separate a TI-tree. One is loading only the itemset tree in memory and leaving all time series counts lists in the hard disk. The other is loading part of the itemset tree with its time series counts lists into memory and leaving the remaining tree branches with their counts lists in the hard disk. Both of them may have their own advantages and disadvantages to be discovered. The development of such a memory management mechanism needs special experiments in advance, and needs to be solved urgently.

7.4.3 Improving the Efficiency of the TI-tree

Although the TI-tree shows efficiency for temporal association rule mining, it can be further improved. The TI-tree uses a descendants avoiding method, which avoids those nodes that can definitely not be frequent during any interval, and avoids those intervals of descendants which can definitely not be long. But the method is not good enough, since more nodes and intervals can be avoided.

For example, in a generated TI-tree, the itemset node ABC can be avoided if its parent AB is not frequent in any interval during the time domain. As the direct ancestor of ABC, all sub-branches of AB can be avoided in this situation to save

time, but since there is no direct relation between BC and ABC in the TI-tree, the current descendants-avoiding method can not be efficient through avoiding ABC if BC is not frequent during any interval. The same problem also happens with interval avoiding. So, it is necessary to improve the TI-tree structure so that the relations between those indirectly-related nodes can be built to create more efficiency.

References

[Abraham and Roddick, 1999]

Abraham, T. and Roddick, J.F. 1999. Incremental Meta-Mining from Large Temporal Data Sets. In *Advances in Database Technologies, Proc. First International Workshop on Data Warehousing and Data Mining*, Lecture Notes in Computer Science 1552, Spdnger-Verlag, Berlin. 41-54.

[Agrawal and Srikant, 1994]

Agrawal, R. and Srikant, R. 1994. Fast Algorithms for Mining Association Rules in Large Databases. In *Proceedings of the 20th International Conference on Very Large Data Bases* (September 12 - 15, 1994). J. B. Bocca, M. Jarke, and C. Zaniolo, Eds. Very Large Data Bases. Morgan Kaufmann Publishers, San Francisco, CA, 487-499.

[Agrawal and Srikant, 1995]

Agrawal, R. and Srikant, R. 1995. Mining Sequential Patterns. In *Proceedings of the Eleventh International Conference on Data Engineering* (March 06 - 10, 1995). P. S. Yu and A. L. Chen, Eds. ICDE. IEEE Computer Society, Washington, DC, 3-14.

[Agrawal et al., 1993]

Agrawal, R., Imieliński, T., and Swami, A. 1993. Database Mining: A Performance Perspective. *IEEE Transactions on Knowledge and Data Engineering* 5, 6 (Dec. 1993), 914-925.

[Agrawal et al., 1993a]

Agrawal, R., Imieliński, T., and Swami, A. 1993. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data* (Washington, D.C., United States, May 25 - 28, 1993). P. Buneman and S. Jajodia, Eds. SIGMOD '93. ACM Press, New York, NY, 207-216.

[Ale and Rossi, 2000]
Ale, J. M. and Rossi, G. H. 2000. An approach to discovering temporal association rules. In *Proceedings of the 2000 ACM Symposium on Applied Computing - Volume 1* (Como, Italy). J. Carroll, E. Damiani, H. Haddad, and D. Oppenheim, Eds. SAC '00. ACM Press, New York, NY, 294-300.

[Anderson, 1982]
Anderson, T. L. 1982. Modeling Time at the Conceptual Level. In *Proceedings of the International Conference on Databases: Improving Usability and Responsiveness*, 273-297, Jerusalem, Israel, (June 1982). Academic Press.

[Antunes and Oliveira, 1998]
Antunes, C. M. and Oliveira, A. L. 1998. Temporal Data Mining: an overview. Proceedings of the Sixth ACM SIGKDD International conference on Knowledge Discovery and Data Mining. (1998).

[Bayardo and Agrawal, 1999]
Bayardo, R. J. and Agrawal, R. 1999. Mining the most interesting rules. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (San Diego, California, United States, August 15 - 18, 1999). KDD '99. ACM Press, New York, NY, 145-154.

[Berry and Linoff, 1997]
Berry, M. and Linoff, G. 1997. Data Mining Techniques: for Marketing, Sales and Customer Support. *John Wiley & Sons Inc.* ISBN: 0471179809

[Chaudhuri, 1998]
Chaudhuri, S. 1998. Data mining and database systems: Where is the intersection? *Bulletin of the Technical Committee on Data Engineering*, 21:4-8, March 1998.

[Chen, 1999]
Chen, X. 1999. Temporal data mining: algorithms language and system for temporal association rules. *Thesis submitted to Manchester Metropolitan University, Department of Computing and Mathematics*. (1999).

[Chen and Petrounias, 1998]
Chen, X. and Petrounias, I. 1998. Language Support for Temporal Data Mining. In *Proceedings of the Second European Symposium on Principles of Data Mining and Knowledge Discovery* (September 23 - 26, 1998). J. M. Zytkow and M. Quafafou, Eds. Lecture Notes In Computer Science, vol. 1510. Springer-Verlag, London, 282-290.

[Chen and Petrounias, 1998a]
Chen, X. and Petrounias, I. 1998. A Framework for Temporal Data Mining. In *Proceedings of the 9th International Conference on Database and Expert Systems Applications* (August 24 - 28, 1998). G. Quirchmayr, E. Schweighofer, and T. J. Bench-Capon, Eds. Lecture Notes In Computer Science, vol. 1460. Springer-Verlag, London, 796-805.

[Chen and Petrounias, 1999]
Chen, X. and Petrounias, I. 1999. Mining Temporal Features in Association Rules. In *Proceedings of the Third European Conference on Principles of Data Mining and Knowledge Discovery* (September 15 - 18, 1999). J. M. Zytkow and J. Rauch, Eds. Lecture Notes In Computer Science, vol. 1704. Springer-Verlag, London, 295-300.

[Chen and Petrounias, 2000]
Chen, X. and Petrounias, I. 2000. An Integrated Query and Mining System for Temporal Association Rules. In *Proceedings of the Second International Conference on Data Warehousing and Knowledge Discovery* (September 04 - 06, 2000). Y. Kambayashi, M. K. Mohania, and A. M. Tjoa, Eds. Lecture Notes In Computer Science, vol. 1874. Springer-Verlag, London, 327-336.

[Chen and Petrounias, 2000a]
Chen, X. and Petrounias, I. 2000. Discovering Temporal Association Rules: Algorithms, Language and System. *Proceedings of the 16th International Conference on Data Engineering* (ICDE'2000), 306 IEEE Computer Society, ISBN: 0769505066

[Chen et al., 1998]
Chen, X. Petrounias, I. and Heathfield, H. Discovering Temporal Association Rules in Temporal Databases. *Proceedings of Workshop on Issues and Applications of Database Technology* (IADT'98), Germany, July 1998.

[Chowdhury, 1991]
Chowdhury, S., Bodemar, G., Haug, P., Babic, A., and Wigertz, O. 1991. Methods of knowledge extraction from a clinical database on liver diseases. *Comput. Biomed. Res.* 24, 6 (Dec. 1991), 530-548.

[Cios et al., 1998]
Cios, K., Pedrycz, W., and Swiniarski, R. 1998. Data Mining Methods for Knowledge Discovery, Kluwer Academic Publishers, 1998

[Clifford and Rao, 1987]
Clifford, J. and Rao, A. 1988. A simple general structure for temporal domains. In *C. Rolland, and M. Leonard, editors, Temporal Aspects of Information Systems*, pages 17--28, Elsevier Science Publishers B.V., IFIP, 1988.

[Clifford and Tansel, 1985]
Clifford, J. and Tansel, A. U. 1985. On an algebra for historical relational databases: two views. In *Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data* (Austin, Texas, United States). SIGMOD '85. ACM Press, New York, NY, 247-265.

[Das et al., 1998]

Das, G., Lin, K.I., Mannila, H., Renganathan, G. and Smyth, P. 1998. Rule Discovery from Time Series. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining* (KDD-98), AAAI Press (1998) 16-22

[Džeroski, 1996]

Džeroski, S. 1996. Inductive logic programming and knowledge discovery in databases. In *Advances in Knowledge Discovery and Data Mining*, U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Eds. American Association for Artificial Intelligence, Menlo Park, CA, 117-152.

[Elder and Pregibon, 1996]

Elder, J. F. and Pregibon, D. 1996. A statistical perspective on knowledge discovery in databases. In *Advances in Knowledge Discovery and Data Mining*, U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, and R. Uthurusamy, Eds. American Association for Artificial Intelligence, Menlo Park, CA, 83-113.

[Etzion et al., 1998]

Etzion, O., Jajodia, S. and Sripada, S. (Editors). 1998. Temporal Databases: Research and Practice. *Springer-Verlag Berlin and Heidelberg GmbH & Co. K.* ISBN: 3540645195

[Fayyad et al., 1996]

Fayyad, U., Piatetsky-Shapiro, G., Smyth, P. and Uthurusamy, R. 1996. Advances in Knowledge Discovery and Data Mining. *The MIT Press*. ISBN: 0262560976

[Fisher, 1987]

Fisher, D.1987. Knowledge acquisition via incremental conceptual clustering. *Machine Learning*. (1987) vol. 2 139-172.

[Giles et al., 2001]
Giles, L. C., Lawrence, S. and Tsoi, A. H. 2001. Noisy time series prediction using recurrent neural network and grammatical inference, *Machine Learning*. (2001) vol. 44 161-183.

[Glymour et al., 1996]
Glymour, C., Madigan, D., Pregibon, D., and Smyth, P. 1996. Statistical inference and data mining. *Commun. ACM* 39, 11 (Nov. 1996), 35-41.

[Han and Kamber, 2001]
Han, J. and Kamber, M. 2000. Data Mining: Concepts and Techniques. *The Morgan Kaufmann Publishers Inc.* (2000). ISBN: 1558604898

[Han et al., 1992]
Han, J., Cai, Y., and Cercone, N. 1992. Knowledge Discovery in Databases: An Attribute-Oriented Approach. In *Proceedings of the 18th International Conference on Very Large Data Bases* (August 23 - 27, 1992). L. Yuan, Ed. Very Large Data Bases. Morgan Kaufmann Publishers, San Francisco, CA, 547-559.

[Han et al., 2000]
Han, J., Pei, J., and Yin, Y. 2000. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data* (Dallas, Texas, United States, May 15 - 18, 2000). SIGMOD '00. ACM Press, New York, NY, 1-12.

[Han et al., 2000a]
Han, J., Pei, J., Mortazavi-Asl, B., Chen, Q., Dayal, U., and Hsu, M. 2000. FreeSpan: frequent pattern-projected sequential pattern mining. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (Boston, Massachusetts, United States, August 20 - 23, 2000). KDD '00. ACM Press, New York, NY, 355-359.

[Hand et al., 2001]
Hand, D., Mannila, H. and Smyth, P. 2001 Principles of Data Mining. *The MIT Press.* (2001).ISBN: 026208290X

[Harinarayan et al., 1996]
Harinarayan, V., Rajaraman, A., and Ullman, J. D. 1996. Implementing data cubes efficiently. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (Montreal, Quebec, Canada, June 04 - 06, 1996). J. Widom, Ed. SIGMOD '96. ACM Press, New York, NY, 205-216.

[Imieliński and Mannila, 1996]
Imieliński, T. and Mannila, H. 1996. A database perspective on knowledge discovery. *Commun. ACM* 39, 11 (Nov. 1996), 58-64.

[Imieliński et al., 1996]
Imieliński, T., Virmani, A., and Abdulghani, A. 1999. DMajor—Application Programming Interface for Database Mining. *Data Min. Knowl. Discov.* 3, 4 (Dec. 1999), 347-372.

[Jensen, 1995]
Jensen, C. S. 1995. Introduction to Temporal Database Research. Proceedings of the 14th International Conference on Object-Oriented and Entity-Relationship Modelling. (1995)

[Keogh and Pazzani, 1998]
Keogh, E. J. and Pazzani, M. J. 1998. An enhanced representation of time series data which allows fast and accurate classification, clustering and relevance feedback. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining.* (KDD-98), ACM Press 239-241.

[Ketterlin, 1997]
Ketterlin, A. 1997. Clustering Sequences of Complex Objects. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining.* (1997), AAAI Press 215-218

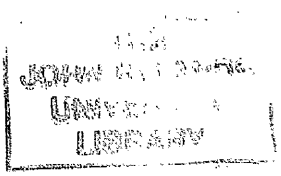
[Lang et al., 1998]
Lang, K. J., Pearlmutter, B. A., and Price, R. A. 1998. Results of the Abbadingo One DFA Learning Competition and a New Evidence-Driven State Merging Algorithm. In *Proceedings of the 4th International Colloquium on Grammatical inference* (July 12 - 14, 1998). V. G. Honavar and G. Slutzki, Eds. Lecture Notes In Computer Science, vol. 1433. Springer-Verlag, London, 1-12.

[Lee et al., 1998]
Lee, J. Y., Elmasri, R., and Won, J. 1998. An integrated temporal data model incorporating time series concept. *Data Knowl. Eng.* 24, 3 (Jan. 1998), 257-276.

[Lee et al., 2001]
Lee, C.H., Lin, C.R., and Chen M.S. 2001. On Mining General Temporal Association Rules in a Publication Database. *Proceedings of ICDM01*, (November 2001), 337-344.

[Li et al., 1999]
Li, Y., Ning, P., Wang, X. S., and Jajodia, S. 2001. Discovering Calendar-Based Temporal Association Rules. In *Proceedings of the Eighth International Symposium on Temporal Representation and Reasoning (Time'01)* (June 14 - 16, 2001). TIME. IEEE Computer Society, Washington, DC, 111.

[Long et al., 1991]
Long, J., Irani, E. and Slagle, J. 1991. Automating the Discovery of Causal Relationships in a Medical Records Database. In *Piatetsky-Shapiro, G. and Frawley, W. (editers) Knowledge Discovery in Databases.* 465-476, The AAAI Press.



[Lu et al., 1996]
Lu, H., Setiono, R., and Liu, H. 1996. Effective Data Mining Using Neural Networks. *IEEE Transactions on Knowledge and Data Engineering* 8, 6 (Dec. 1996), 957-961.

[Lu et al., 1998]
Lu, H., Han, J. and Feng, L. 1998. Stock movement prediction and n-dimensional inter-transaction association rules. In *Proceedings of ACM SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery*. (1998), 12:1-12:7

[Mannila and Toivonen, 1996]
Mannila, H. and Toivonen, H. On an algorithm for finding all interesting sentences. In *Cybernetics and Systems, Volume II.* The Thirteenth European Meeting on Cybernetics and Systems Research, 973-978.

[Mannila et al., 1999]
Mannila, H., Pavlov, D. and Smyth, P. 1999 Predictions with local patterns using crossentropy. In *Proceedings of Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, (1999), ACM Press 357-361.

[Megiddo and Srikant, 1999]
Megiddo, N. and Srikant, R. 1998. Discovering predictive association rules. *Knowledge Discovery and Data Mining* (KDD-98). 274-278.

[Muggleton and De Raedt, 1994]
Muggleton, S. and De Raedt, L. 1994. Inductive logic programming : Theory and methods. *Journal of Logic Programming*, (19, 20) 629-679, 1994.

[Özden et al., 1998]
Özden, B., Ramaswamy, S., and Silberschatz, A. 1998. Cyclic Association Rules. In *Proceedings of the Fourteenth International Conference on Data Engineering* (February 23 - 27, 1998). ICDE. IEEE Computer Society, Washington, DC, 412-421.

[Quinlan, 1986]
Quinlan, J. R. 2003. Induction of Decision Trees. *Mach. Learn.* 1, 1 (Feb. 2003), 81-106.

[Quinlan 1993]
Quinlan, J. R. 1993 *C4.5: Programs for Machine Learning*. Morgan Kaufmann Publishers Inc.

[Ramaswamy et al., 1998]
Ramaswamy, S., Mahajan, S., and Silberschatz, A. 1998. On the Discovery of Interesting Patterns in Association Rules. In *Proceedings of the 24th International Conference on Very Large Data Bases* (August 24 - 27, 1998). A. Gupta, O. Shmueli, and J. Widom, Eds. Very Large Data Bases. Morgan Kaufmann Publishers, San Francisco, CA, 368-379.

[Sarawagi et al., 1998]
Sarawagi, S., Thomas, S., and Agrawal, R. 2000. Integrating Association Rule Mining with Relational Database Systems: Alternatives and Implications. *Data Min. Knowl. Discov.* 4, 2-3 (Jul. 2000), 89-125.

[Saraee and Theodoulidis, 1995]
Saraee, M. H. and Theodoulidis, B. 1995. Knowledge Discovery in Temporal Databases. Proceedings of the DOOD'95 Post-Conference Workshops on Integration of KDOOD and TDOOD, Singapore, National University of Singapore (NUS), (1995) 17-22

[Savasere et al., 1995]
Savasere, A., Omiecinski, E. and Navathe, S. 1995. An Efficient Algorithm for mining Association Rules in Large Databases. In *Proceedings of the 21st VLDB Conference* Zurich, Swizerland. (1995), 423-443.

[Smyth, 1997]
Smyth, P. 1997. Clustering sequences using hidden Markov models. In *Mozer, M., Jordan, M. and Petsche, T., editors, Advances in Neural Information Processing Systems*. (1997), vol. 9 648-654, The MIT Press.

[Smyth, 1999]
Smyth, P. Probabilistic model-based clustering of multivariate and sequential data. In *Proceedings of Artificial Intelligence and Statistics* (1999), 299-304. Morgan Kaufman

[Snodgrass, 1987]
Snodgrass, R. 1987. The temporal query language TQuel. *ACM Trans. Database Syst.* 12, 2 (Jun. 1987), 247-298.

[Srikant and Agrawal, 1996]
Srikant, R. and Agrawal, R. 1996. Mining quantitative association rules in large relational tables. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data* (Montreal, Quebec, Canada, June 04 - 06, 1996). J. Widom, Ed. SIGMOD '96. ACM Press, New York, NY, 1-12.

[Thuraisingham, 1999]
Thuraisingham, B. 1998. Data Mining: Technologies, Techniques, Tools, and Trends. *CRC Press*. ISBN: 0849318157

[Toivonen, 1996]
Toivonen, H. 1996. Sampling Large Databases for Association Rules. In *Proceedings of the 22nd VLDB Conference*. Mumbai (Bombay), India (1996), 134-145.

[Wang and Tjortjis, 2004]
Wang, C. and Tjortjis, C. 2004. PRICES: An Efficient Algorithm for Mining Association Rules. In *Proceedings of 5th IDEAL*, Lecture Notes Computer Science, (2004), Vol. 3177, 352-358, Springer-Verlag Berlin Heidelberg.

[Weigend and Gershenfeld, 1994]
Weigend, A.S. and Gershenfeld, N.A. 1994. Times Series Prediction: Forecasting the future and Understanding the Past, *Addison Wesley Publishing Company*. (1994), ISBN: 0201626020

[Zaniolo et al., 1997]
Zaniolo, C., Ceri, S., Faloutsos, C., Snodgrass, R. T., Subrahmanian, V. S., and Zicari, R. 1997 *Advanced Database Systems*. Morgan Kaufmann Publishers Inc.

Appendix A: Program IARMiner

Class IARMiner.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import com.ms.wfc.io.*;
import com.ms.wfc.util.*;

public class IARMiner extends Form
{
  public IARMiner()
  {
    initForm();
    initial();
  }

  public void dispose()
  {
    super.dispose();
    components.dispose();
  }

  float minimumSupport = 0;          //minimum support threshold from user input
  int minimumIntervalLength = 0;  //minimum interval length threshold from user input
  String temporalDatabaseFileName = null;  //database file to be mined from user input
  TextReader reader = null;          //instance of TextReader class, used to open database file
  long transactionID;
  long date;
  int itemNumber;          //used to present the itemNumber field of each transaction
  ILSList iLSList;          //the root node of data structure

  void step1IARMiner()
  {
    //Check begin and end point of time domain
    reader = new TextReader(temporalDatabaseFileName);
    String nextLine = reader.readLine();
    long domainStart = java.lang.Long.parseLong(nextLine.substring(6,10).trim());
    long domainEnd = 0;
    while (nextLine != null)
    {
      domainEnd = java.lang.Long.parseLong(nextLine.substring(6,10).trim());
      nextLine = reader.readLine();
    }
    reader.close();
    //End check, result stored in long domainStart and long domianEnd

    //Begin of interval large itemset list data structure
    iLSList = new ILSList();
    iLSList.setTimeDomainStartPoint(domainStart);
    iLSList.setTimeDomainEndPoint(domainEnd);
    IntervalList intervalList = new IntervalList();
    intervalList.setIntervalLength(minimumIntervalLength);
    iLSList.enqueueIntervalList(intervalList);
    //End of interval large itemset list data structure

    //Minimum time interval control for first step
    long minimumTimeIntervalStartPoint = 0;
    long minimumTimeIntervalEndPoint = 0;
    minimumTimeIntervalStartPoint = domainStart;
    minimumTimeIntervalEndPoint = minimumTimeIntervalStartPoint + minimumIntervalLength - 1;
    while (minimumTimeIntervalEndPoint <= domainEnd)
    {
      labelStart.setText(String.valueOf(minimumTimeIntervalStartPoint));//can be deleted to save time
      labelEnd.setText(String.valueOf(minimumTimeIntervalEndPoint));// can be deleted to save time
```

```
    //Normal association rule mining process
    IntervalNode intervalNode = new IntervalNode();
    intervalNode.setStartTimePoint(minimumTimeIntervalStartPoint);
    intervalNode.setEndTimePoint(minimumTimeIntervalEndPoint);
    intervalList.enqueueIntervalNode(intervalNode);
    apriori(intervalNode, minimumTimeIntervalStartPoint, minimumTimeIntervalEndPoint);
    //End of normal association rule mining process

    minimumTimeIntervalStartPoint ++;
    minimumTimeIntervalEndPoint ++;
  }
  //End of minimum time interval control for first step
}

void apriori(IntervalNode iNode, long start, long end)
{
  long numberOfTransaction = 0;
  long tid = 9999;

  //Search for begin point
  reader = new TextReader(temporalDatabaseFileName);
  String nextLine = reader.readLine();
  while (java.lang.Long.parseLong(nextLine.substring(6,10).trim()) < start)
  {
    nextLine = reader.readLine();
  }
  //End of searching begin point

  //Start of searching 1-large item set of apriori
  boolean added = false;
  if ((java.lang.Long.parseLong(nextLine.substring(6,10).trim()) <= end) && (nextLine != null))
  {
    while (java.lang.Long.parseLong(nextLine.substring(6,10).trim()) <= end)
    {
      itemNumber = java.lang.Integer.parseInt(nextLine.substring(10,15).trim());
      added = false;
      labelLargeItemSet.setText(String.valueOf(itemNumber));//can be deleted to save time
      if (iNode.getEmptyLargeItemsetList())
      {
        LargeItemset largeItemset = new LargeItemset();
        largeItemset.setLargeItemsetSize(1);
        largeItemset.setCount(1);
        iNode.enqueueLargeItemset(largeItemset);
        LargeItem largeItem = new LargeItem();
        largeItem.setLargeItemNumber(itemNumber);
        largeItemset.enqueueLargeItem(largeItem);
        largeItemset = null;
        largeItem = null;
        added = true;
      }
      else
      {
        for (int i = 0; i < iNode.getAmountOfLargeItemset(); i++)
        {
          LargeItemset lis1 = iNode.getLargeItemset(i);
          LargeItem li1 = lis1.getLargeItem(0);
          if (itemNumber < li1.getLargeItemNumber())
          {
            LargeItemset largeItemset = new LargeItemset();
            largeItemset.setLargeItemsetSize(1);
            largeItemset.setCount(1);
            iNode.insertLargeItemset(i,largeItemset);
            LargeItem largeItem = new LargeItem();
            largeItem.setLargeItemNumber(itemNumber);
            largeItemset.enqueueLargeItem(largeItem);
            largeItemset = null;
            largeItem = null;
            lis1 = null;
```

```
            li1 = null;
            added = true;
            break;
          }
          if (itemNumber > li1.getLargeItemNumber())
            continue;
          if (itemNumber == li1.getLargeItemNumber())
          {
            lis1.increaseCount();
            lis1 = null;
            li1 = null;
            added = true;
            break;
          }
          lis1 = null;
          li1 = null;
        }
        if (added == false)
        {
          LargeItemset largeItemset = new LargeItemset();
          largeItemset.setLargeItemsetSize(1);
          largeItemset.setCount(1);
          iNode.enqueueLargeItemset(largeItemset);
          LargeItem largeItem = new LargeItem();
          largeItem.setLargeItemNumber(itemNumber);
          largeItemset.enqueueLargeItem(largeItem);
          largeItemset = null;
          largeItem = null;
        }
      }
      if (tid != java.lang.Long.parseLong(nextLine.substring(0,6).trim()))
      {
        tid = java.lang.Long.parseLong(nextLine.substring(0,6).trim());
        numberOfTransaction ++;
        labelTID.setText(String.valueOf(tid));//can be deleted to save time
      }
      nextLine = reader.readLine();
      if (nextLine == null)
        break;
    }
  }
  //End of searching 1-large item set of apriori

  //Close
  reader.close();
  //End of close

  //Begin of searching 2+-large item set of apriori
  int currentLargestItemsetSize = 1;
  long minimumCount = (long)(numberOfTransaction * minimumSupport) + 1;
  LargeItemset lis2 = null;
  for (int i = 0; i < iNode.getAmountOfLargeItemset();)
  {
    lis2 = iNode.getLargeItemset(i);
    if (lis2.getCount() < minimumCount)
    {
      lis2 = null;
      iNode.removeLargeItemset(i);
      continue;
    }
    else
    {
      lis2 = null;
      i++;
    }
  }
  iNode.trimLargeItemsetListToSize();
  boolean goOn = true;
  while ((goOn == true) && (iNode.getAmountOfLargeItemset() > 1))
```

```
    {
      currentLargestItemsetSize ++;
      goOn = aprioriGenAndCount(iNode, currentLargestItemsetSize, start, end, minimumCount);
    }
    //End of searching 2+-large item set of apriori
}

boolean aprioriGenAndCount(IntervalNode inode, int cLIS, long start, long end, long minimumCount)
{
    //Apriori-Gen
    int pLIS = cLIS - 1;
    int pLIA = 0;
    int cLIA = 0;
    LargeItemset lis3 = null;
    LargeItemset lis4 = null;
    LargeItem li3 = null;
    LargeItem li4 = null;
    for (int i = 0; i < inode.getAmountOfLargeItemset(); i++)
    {
      lis3 = inode.getLargeItemset(i);
      if (lis3.getLargeItemsetSize() == pLIS)
        pLIA ++;
      lis3 = null;
    }
    if (pLIA < 2)
      return false;
    if (pLIS >= 2)
    {
      int size = inode.getAmountOfLargeItemset();
      for (int i = 0; i < size; i++)
      {
        lis3 = inode.getLargeItemset(i);
        if (lis3.getLargeItemsetSize() != pLIS)
        {
          lis3 = null;
          continue;
        }
        for (int j = i + 1; j < size; j++)
        {
          lis4 = inode.getLargeItemset(j);
          if (lis4.getLargeItemsetSize() != pLIS)
          {
            lis4 = null;
            continue;
          }
          boolean joinAble = true;
          for (int k = 0; k < pLIS - 1; k++)
          {
            li3 = lis3.getLargeItem(k + 1);
            li4 = lis4.getLargeItem(k);
            if (li3.getLargeItemNumber() != li4.getLargeItemNumber())
            {
              joinAble = false;
              li3 = null;
              li4 = null;
              break;
            }
            li3 = null;
            li4 = null;
          }
          if (joinAble)
          {
            LargeItemset lis = new LargeItemset();
            lis.setLargeItemsetSize(cLIS);
            lis.setCount(0);
            inode.enqueueLargeItemset(lis);
            LargeItem li = null;
            for (int k = 0; k < pLIS; k ++)
            {
```

```
            li3 = lis3.getLargeItem(k);
            li = new LargeItem();
            li.setLargeItemNumber(li3.getLargeItemNumber());
            lis.enqueueLargeItem(li);
            li = null;
            li3 = null;
          }
          li4 = lis4.getLargeItem(pLIS - 1);
          li = new LargeItem();
          li.setLargeItemNumber(li4.getLargeItemNumber());
          lis.enqueueLargeItem(li);
          li = null;
          li4 = null;
        }
        lis4 = null;
      }
      lis3 = null;
    }
  }
  else
  {
    if ((pLIS == 1) && (inode.getAmountOfLargeItemset() >= 2))
    {
      LargeItemset lis = null;
      LargeItem li = null;
      int size1 = inode.getAmountOfLargeItemset();
      for (int i = 0; i < size1 - 1; i++)
      {
        lis3 = inode.getLargeItemset(i);
        li3 = lis3.getLargeItem(0);
        for (int j = i + 1; j < size1; j++)
        {
          lis4 = inode.getLargeItemset(j);
          li4 = lis4.getLargeItem(0);
          lis = new LargeItemset();
          lis.setLargeItemsetSize(2);
          li = new LargeItem();
          inode.enqueueLargeItemset(lis);
          li.setLargeItemNumber(li3.getLargeItemNumber());
          lis.enqueueLargeItem(li);
          li = null;
          li = new LargeItem();
          li.setLargeItemNumber(li4.getLargeItemNumber());
          lis.enqueueLargeItem(li);
          li = null;
          lis = null;
        }
      }
    }
    else
      return false;
  }
  //End of apriori-Gen

  //Count
  long cCount = 0;
  long tTID = 9999;
  int tItemNumber = 9999;
  for (int i = 0; i < inode.getAmountOfLargeItemset(); i++)
  {
    lis3 = inode.getLargeItemset(i);
    if (lis3.getLargeItemsetSize() == cLIS)
      cLIA ++;
    lis3 = null;
  }
  if (cLIA < 1)
    return false;
  for (int i = 0; i < inode.getAmountOfLargeItemset(); i++)
  {
```

```
lis3 = inode.getLargeItemset(i);
if (lis3.getLargeItemsetSize() == cLIS)
{
  String itse = "-";                                                    //can be deleted to save time
  for (int m = 0; m < cLIS; m++)                                        //can be deleted to save time
  {                                                                     //can be deleted to save time
    li4 = lis3.getLargeItem(m);                                         //can be deleted to save time
    itse = itse + " " + String.valueOf(li4.getLargeItemNumber());       //can be deleted to save time
  }                                                                     //can be deleted to save time
  labelLargeItemSet.setText(itse);                                      //can be deleted to save time

  reader = new TextReader(temporalDatabaseFileName);
  String nextLine = reader.readLine();
  while (java.lang.Long.parseLong(nextLine.substring(6,10).trim()) < start)
  {
    nextLine = reader.readLine();
  }

  if ((java.lang.Long.parseLong(nextLine.substring(6,10).trim()) <= end) && (nextLine != null))
  {
    tTID = java.lang.Long.parseLong(nextLine.substring(0,6).trim());
    labelTID.setText(String.valueOf(tTID));                             //can be deleted to save time
    while (java.lang.Long.parseLong(nextLine.substring(6,10).trim()) <= end)
    {
      tItemNumber = java.lang.Integer.parseInt(nextLine.substring(10,15).trim());
      if (tTID == java.lang.Long.parseLong(nextLine.substring(0,6).trim()))
      {
        for (int j = 0; j < lis3.getAmountOfLargeItem(); j++)
        {
          li3 = lis3.getLargeItem(j);
          if (li3.getLargeItemNumber() == tItemNumber)
            cCount ++;
        }
      }
      else
      {
        if (cCount >= cLIS)
          lis3.increaseCount();
        cCount = 0;
        tTID = java.lang.Long.parseLong(nextLine.substring(0,6).trim());
        labelTID.setText(String.valueOf(tTID));//can be deleted to save time
        for (int j = 0; j < lis3.getAmountOfLargeItem(); j++)
        {
          li3 = lis3.getLargeItem(j);
          if (li3.getLargeItemNumber() == tItemNumber)
            cCount ++;
        }
      }
      nextLine = reader.readLine();
      if (nextLine == null)
        break;
    }
  }
  reader.close();
}
else
  continue;
lis3 = null;
}

LargeItemset lis2 = null;
for (int i = 0; i < inode.getAmountOfLargeItemset();)
{
  lis2 = inode.getLargeItemset(i);
  if (lis2.getCount() < minimumCount)
  {
    lis2 = null;
    inode.removeLargeItemset(i);
    continue;
```

```
      }
      else
      {
        lis2 = null;
        i++;
      }
    }
    inode.trimLargeItemsetListToSize();

    return true;
    //End of Count
  }

  void step2IARMiner()
  {
    IntervalList iL1 = null;
    iL1 = iLSList.getIntervalList(iLSList.getAmountOfIntervalList() - 1);
    while (iL1.getAmountOfIntervalNode() > 1)
    {
      //Combine
      IntervalList iL = new IntervalList();
      iL.setIntervalLength(iL1.getIntervalLength() + 1);
      iLSList.enqueueIntervalList(iL);

      IntervalList iL2 = null;
      iL2 = iL1;
      IntervalNode iN1 = null;
      IntervalNode iN2 = null;
      IntervalNode iN = null;
      for (int i = 0; i < iL2.getAmountOfIntervalNode() - 1; i++)
      {
        iN1 = iL2.getIntervalNode(i);
        iN2 = iL2.getIntervalNode(i + 1);
        if (((iN1.getStartTimePoint() + 1) == (iN2.getStartTimePoint())) && ((iN1.getEndTimePoint()
+ 1) == (iN2.getEndTimePoint())))
        {
          iN = new IntervalNode();
          iN.setStartTimePoint(iN1.getStartTimePoint());
          iN.setEndTimePoint(iN2.getEndTimePoint());
          iL.enqueueIntervalNode(iN);

          LargeItemset lIS1 = null;
          LargeItemset lIS2 = null;
          LargeItemset lIS = null;
          LargeItem lI1 = null;
          LargeItem lI2 = null;
          LargeItem lI = null;
          for (int j = 0; j < iN1.getAmountOfLargeItemset(); j++)
          {
            for (int k = 0; k < iN2.getAmountOfLargeItemset(); k++)
            {
              lIS1 = iN1.getLargeItemset(j);
              lIS2 = iN2.getLargeItemset(k);
              if (lIS1.getAmountOfLargeItem() == lIS2.getAmountOfLargeItem())
              {
                boolean same = true;
                for (int l = 0; l < lIS1.getAmountOfLargeItem(); l++)//check exactly same
                {
                  lI1 = lIS1.getLargeItem(l);
                  boolean same1 = false;
                  for (int m = 0; m < lIS2.getAmountOfLargeItem(); m++)
                  {
                    lI2 = lIS2.getLargeItem(m);
                    if (lI1.getLargeItemNumber() == lI2.getLargeItemNumber())
                      same1 = same1 | true;
                    else
                      same1 = same1 | false;
                    lI2 = null;
                  }
```

```
            if (same1 == true)
              same = same & true;
            else
              same = same & false;
            lI1 = null;
          }//End of check
          if (same == true)
          {
            lIS = new LargeItemset();
            iN.enqueueLargeItemset(lIS);
            for (int n = 0; n < lIS1.getAmountOfLargeItem(); n++)
            {
              lI = new LargeItem();
              lI1 = lIS1.getLargeItem(n);
              lI.setLargeItemNumber(lI1.getLargeItemNumber());
              lIS.enqueueLargeItem(lI);
              lI = null;
              lI1 = null;
            }
            lIS = null;
          }
        }
        lIS1 = null;
        lIS2 = null;
      }
    }
  }
  iN2 = null;
  iN1 = null;
}

iN2 = null;
iN1 = null;
//End of combine

//Count
IntervalNode cIN = null;
LargeItemset cLIS = null;
LargeItem cLI = null;
long tTID = 9999;
int tItemNumber = 9999;
long cCount = 0;
long mCount = 0;
for (int i = 0; i < iL.getAmountOfIntervalNode(); i++)
{
  cIN = iL.getIntervalNode(i);
  labelStart.setText(String.valueOf(cIN.getStartTimePoint()));     //can be deleted to save time
  labelEnd.setText(String.valueOf(cIN.getEndTimePoint()));         //can be deleted to save time
  for (int j = 0; j < cIN.getAmountOfLargeItemset(); j++)
  {
    cLIS = cIN.getLargeItemset(j);
                                                                   //can be deleted to save time
    String itse = "-";                                             //can be deleted to save time
    for (int m = 0; m < cLIS.getAmountOfLargeItem(); m++)          //can be deleted to save time
    {                                                              //can be deleted to save time
      cLI = cLIS.getLargeItem(m);                                  //can be deleted to save time
      itse = itse + " " + String.valueOf(cLI.getLargeItemNumber());//can be deleted to save time
      cLI = null;                                                  //can be deleted to save time
    }                                                              //can be deleted to save time
    labelLargeItemSet.setText(itse);                               //can be deleted to save time
                                                                   //can be deleted to save time

    reader = new TextReader(temporalDatabaseFileName);
    String nextLine = reader.readLine();
    while (java.lang.Long.parseLong(nextLine.substring(6,10).trim()) < cIN.getStartTimePoint())
    {
      nextLine = reader.readLine();
    }
```

```
        if ((java.lang.Long.parseLong(nextLine.substring(6,10).trim()) <= cIN.getEndTimePoint())
&& (nextLine != null))
        {
          tTID = java.lang.Long.parseLong(nextLine.substring(0,6).trim());
          mCount = 0;
          labelTID.setText(String.valueOf(tTID));                    //can be deleted to save time
          while (java.lang.Long.parseLong(nextLine.substring(6,10).trim()) <=
cIN.getEndTimePoint())
          {
            tItemNumber = java.lang.Integer.parseInt(nextLine.substring(10,15).trim());
            if (tTID == java.lang.Long.parseLong(nextLine.substring(0,6).trim()))
            {
              for (int k = 0; k < cLIS.getAmountOfLargeItem(); k++)
              {
                cLI = cLIS.getLargeItem(k);
                if (cLI.getLargeItemNumber() == tItemNumber)
                  cCount ++;
                cLI = null;
              }
            }
            else
            {
              if (cCount >= cLIS.getAmountOfLargeItem())
                cLIS.increaseCount();
              cCount = 0;
              mCount ++;          //for count transactions
              tTID = java.lang.Long.parseLong(nextLine.substring(0,6).trim());
              labelTID.setText(String.valueOf(tTID));//can be deleted to save time
              for (int l = 0; l < cLIS.getAmountOfLargeItem(); l++)
              {
                cLI = cLIS.getLargeItem(l);
                if (cLI.getLargeItemNumber() == tItemNumber)
                  cCount ++;
                cLI = null;
              }
            }
            nextLine = reader.readLine();
            if (nextLine == null)
              break;
          }
        }

        reader.close();
        cLIS = null;
      }
      cIN = null;
    }

    //trim
    long mC = (long)(mCount * minimumSupport) + 1;
    for (int i = 0; i < iL.getAmountOfIntervalNode();)
    {
      cIN = iL.getIntervalNode(i);
      for (int j = 0; j < cIN.getAmountOfLargeItemset();)
      {
        cLIS = cIN.getLargeItemset(j);
        if (cLIS.getCount() < mC)
        {
          cLIS = null;
          cIN.removeLargeItemset(j);
          continue;
        }
        else
        {
          cLIS = null;
          j++;
        }
        cLIS = null;
      }
```

```
        if (cIN.getAmountOfLargeItemset() < 1)
        {
          cIN = null;
          iL.removeIntervalNode(i);
          continue;
        }
        else
        {
          cIN = null;
          i++;
        }
        cIN = null;
      }
      //end of trim

      iL2 = null;
      //End of count
      iL1 = null;
      iL1 = iLSList.getIntervalList(iLSList.getAmountOfIntervalList() - 1);
    }
  }

  void initial()
  {
    situationCheck();
  }
  void situationPreparing()
  {
    buttonAllocationDataFile.setEnabled(true);
    comboBoxMinimumSupport.setEnabled(true);
    comboBoxMinimumIntervalLength.setEnabled(true);
    buttonRun.setEnabled(false);
  }
  void situationPrepared()
  {
    buttonAllocationDataFile.setEnabled(false);
    comboBoxMinimumSupport.setEnabled(false);
    comboBoxMinimumIntervalLength.setEnabled(false);
    buttonRun.setEnabled(true);
  }
  void situationRunning()
  {
    buttonAllocationDataFile.setEnabled(false);
    comboBoxMinimumSupport.setEnabled(false);
    comboBoxMinimumIntervalLength.setEnabled(false);
    buttonRun.setEnabled(false);
  }
  void situationRunned()
  {
    buttonAllocationDataFile.setEnabled(true);
    comboBoxMinimumSupport.setEnabled(true);
    comboBoxMinimumIntervalLength.setEnabled(true);
    buttonRun.setEnabled(false);
  }
  void situationCheck()
  {
    if ((comboBoxMinimumSupport.getSelectedIndex() != -1) &&
(comboBoxMinimumIntervalLength.getSelectedIndex() != -1) &&
editTargetDataFileName.getText().length() >= 1)
    {
      situationPrepared();
    }
    else
    {
      situationPreparing();
    }
  }

  private void comboBoxMinimumSupport_selectedIndexChanged(Object source, Event e)
```

```
  {
    situationCheck();
  }

  private void comboBoxMinimumIntervalLength_selectedIndexChanged(Object source, Event e)
  {
    situationCheck();
  }

  Time time1;
  Time time2;
  Time time3;
  long timeOfStep1;
  long timeOfStep2;
  long timeOfIARSeeker;
  float tOS1;
  float tOS2;
  float tOIARSeeker;

  private void buttonRun_click(Object source, Event e)
  {
    buttonRun.setEnabled(false);
    minimumSupport =
java.lang.Integer.parseInt(comboBoxMinimumSupport.getSelectedItem().toString());
    minimumSupport = minimumSupport / 100;
    minimumIntervalLength =
java.lang.Integer.parseInt(comboBoxMinimumIntervalLength.getSelectedItem().toString());
    MessageBox.show(String.valueOf(minimumSupport) + String.valueOf(minimumIntervalLength));
    time1 = new Time();
    step1IARMiner();
    time2 = new Time();
    step2IARMiner();
    time3 = new Time();
    MessageBox.show("FINISHED!");
    timeOfStep1 = time2.toLong() - time1.toLong();
    timeOfStep2 = time3.toLong() - time2.toLong();
    timeOfIARSeeker = time3.toLong() - time1.toLong();
    tOS1 = (float)timeOfStep1 / (float)10000000;
    tOS2 = (float)timeOfStep2 / (float)10000000;
    tOIARSeeker = (float)timeOfIARSeeker / (float)10000000;
    MessageBox.show("Step1 = " + String.valueOf(tOS1) + " sec");
    MessageBox.show("Step2 = " + String.valueOf(tOS2) + " sec");
    MessageBox.show("IARSeeker = " + String.valueOf(tOIARSeeker) + " sec");
    buttonRun.setEnabled(true);

    //this is new result section
    Result result = new Result(iLSList);
    result.setEnabled(true);
    result.setVisible(true);
    //end
  }

  private void buttonAllocationDataFile_click(Object source, Event e)
  {
    openFileDialog1.setDefaultExt("tdf");
    openFileDialog1.setFilter("Temporal Database file (*.tdf)|*.tdf");
    openFileDialog1.setFilterIndex(1);
    openFileDialog1.setInitialDir("C:\\Documents and Settings\\ Documents");
    int dlgResult = openFileDialog1.showDialog();
    if (dlgResult == DialogResult.OK)
    {
      temporalDatabaseFileName = openFileDialog1.getFileName();
      editTargetDataFileName.setText(temporalDatabaseFileName);
    }
    situationCheck();
  }

  /**
   * NOTE: The following code is required by the Visual J++ form
```

```
 * designer.  It can be modified using the form editor.  Do not
 * modify it using the code editor.
 */
Container components = new Container();
Label labelAnancement = new Label();
OpenFileDialog openFileDialog1 = new OpenFileDialog();
Label labelTargetDataset = new Label();
Edit editTargetDataFileName = new Edit();
Button buttonAllocationDataFile = new Button();
ComboBox comboBoxMinimumSupport = new ComboBox();
Label labelMinimumSupport = new Label();
Label labelMinimumIntervalLength = new Label();
ComboBox comboBoxMinimumIntervalLength = new ComboBox();
Button buttonRun = new Button();
Label labelMonitor = new Label();
Label labelNumber = new Label();
Label labelTransactionID = new Label();
Label labelTID = new Label();
Label labelDate = new Label();
Label labelID = new Label();
Label labelItemNumber = new Label();
Label labelIN = new Label();
Label labelIAR = new Label();
Label labelTI = new Label();
Label labelS = new Label();
Label labelE = new Label();
Label labelLS = new Label();
Label labelStart = new Label();
Label labelEnd = new Label();
Label label_ = new Label();
Label labelLargeItemSet = new Label();

private void initForm()
{
  this.setText("IARMinerNew");
  this.setAutoScaleBaseSize(new Point(5, 13));
  this.setClientSize(new Point(632, 446));

  labelAnancement.setLocation(new Point(16, 16));
  labelAnancement.setSize(new Point(600, 16));
  labelAnancement.setTabIndex(0);
  labelAnancement.setTabStop(false);
  labelAnancement.setText("This IARMiner Program is only for evaluation of the IARMiner
Algorithm rather than any other real application.");
  labelAnancement.setTextAlign(HorizontalAlignment.CENTER);

  /* @designTimeOnly openFileDialog1.setLocation(new Point(408, 112)); */

  labelTargetDataset.setLocation(new Point(8, 48));
  labelTargetDataset.setSize(new Point(80, 16));
  labelTargetDataset.setTabIndex(1);
  labelTargetDataset.setTabStop(false);
  labelTargetDataset.setText("Target Data Set:");

  editTargetDataFileName.setBackColor(Color.CONTROL);
  editTargetDataFileName.setEnabled(false);
  editTargetDataFileName.setLocation(new Point(96, 48));
  editTargetDataFileName.setSize(new Point(400, 48));
  editTargetDataFileName.setTabIndex(2);
  editTargetDataFileName.setText("");
  editTargetDataFileName.setBorderStyle(BorderStyle.FIXED_SINGLE);
  editTargetDataFileName.setMultiline(true);

  buttonAllocationDataFile.setLocation(new Point(504, 48));
  buttonAllocationDataFile.setSize(new Point(112, 24));
  buttonAllocationDataFile.setTabIndex(3);
  buttonAllocationDataFile.setText("Allocation Data File");
  buttonAllocationDataFile.addOnClick(new EventHandler(this.buttonAllocationDataFile_click));
```

```
    comboBoxMinimumSupport.setLocation(new Point(104, 112));
    comboBoxMinimumSupport.setSize(new Point(64, 21));
    comboBoxMinimumSupport.setTabIndex(4);
    comboBoxMinimumSupport.setText("Select");
    comboBoxMinimumSupport.setItems(new Object[] {"5", "10", "15", "20", "25", "30", "35", "40",
"45", "50", "55", "60", "65", "70", "75", "80", "85", "90", "95"});
    comboBoxMinimumSupport.addOnSelectedIndexChanged(new
EventHandler(this.comboBoxMinimumSupport_selectedIndexChanged));

    labelMinimumSupport.setLocation(new Point(8, 112));
    labelMinimumSupport.setSize(new Point(96, 16));
    labelMinimumSupport.setTabIndex(5);
    labelMinimumSupport.setTabStop(false);
    labelMinimumSupport.setText("Minimum Support %:");

    labelMinimumIntervalLength.setLocation(new Point(176, 112));
    labelMinimumIntervalLength.setSize(new Point(120, 16));
    labelMinimumIntervalLength.setTabIndex(6);
    labelMinimumIntervalLength.setTabStop(false);
    labelMinimumIntervalLength.setText("Minimum Interval Length:");

    comboBoxMinimumIntervalLength.setLocation(new Point(296, 112));
    comboBoxMinimumIntervalLength.setSize(new Point(64, 21));
    comboBoxMinimumIntervalLength.setTabIndex(7);
    comboBoxMinimumIntervalLength.setText("Select");
    comboBoxMinimumIntervalLength.setItems(new Object[] {"1", "2", "3", "4", "5", "6", "7", "8",
"9", "10", "11", "12", "13", "14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26",
"27", "28", "29", "30", "31", "33", "35", "37", "39", "41", "43", "45", "47", "49", "51", "53", "55", "57",
"59", "61", "63", "65", "67", "69", "71", "73", "75", "77", "79", "81", "83", "85", "87", "89"});
    comboBoxMinimumIntervalLength.addOnSelectedIndexChanged(new
EventHandler(this.comboBoxMinimumIntervalLength_selectedIndexChanged));

    buttonRun.setLocation(new Point(528, 104));
    buttonRun.setSize(new Point(75, 23));
    buttonRun.setTabIndex(8);
    buttonRun.setText("Run");
    buttonRun.addOnClick(new EventHandler(this.buttonRun_click));

    labelMonitor.setLocation(new Point(8, 144));
    labelMonitor.setSize(new Point(40, 23));
    labelMonitor.setTabIndex(9);
    labelMonitor.setTabStop(false);
    labelMonitor.setText("Monitor:");

    labelNumber.setLocation(new Point(48, 144));
    labelNumber.setSize(new Point(100, 23));
    labelNumber.setTabIndex(10);
    labelNumber.setTabStop(false);
    labelNumber.setText("");

    labelTransactionID.setLocation(new Point(8, 240));
    labelTransactionID.setSize(new Point(72, 23));
    labelTransactionID.setTabIndex(11);
    labelTransactionID.setTabStop(false);
    labelTransactionID.setText("TransactionID:");

    labelTID.setLocation(new Point(80, 240));
    labelTID.setSize(new Point(100, 23));
    labelTID.setTabIndex(12);
    labelTID.setTabStop(false);
    labelTID.setText("");

    labelDate.setLocation(new Point(192, 240));
    labelDate.setSize(new Point(32, 23));
    labelDate.setTabIndex(13);
    labelDate.setTabStop(false);
    labelDate.setText("Date:");

    labelD.setLocation(new Point(224, 240));
```

```
labelD.setSize(new Point(100, 23));
labelD.setTabIndex(14);
labelD.setTabStop(false);
labelD.setText("");

labelItemNumber.setLocation(new Point(336, 240));
labelItemNumber.setSize(new Point(64, 23));
labelItemNumber.setTabIndex(15);
labelItemNumber.setTabStop(false);
labelItemNumber.setText("ItemNumber:");

labelIN.setLocation(new Point(400, 240));
labelIN.setSize(new Point(100, 23));
labelIN.setTabIndex(16);
labelIN.setTabStop(false);
labelIN.setText("");

labelIAR.setLocation(new Point(8, 296));
labelIAR.setSize(new Point(120, 23));
labelIAR.setTabIndex(17);
labelIAR.setTabStop(false);
labelIAR.setText("Interval Association Rule:");

labelTI.setLocation(new Point(8, 328));
labelTI.setSize(new Point(72, 23));
labelTI.setTabIndex(18);
labelTI.setTabStop(false);
labelTI.setText("Time Interval:");

labelS.setLocation(new Point(80, 328));
labelS.setSize(new Point(32, 23));
labelS.setTabIndex(19);
labelS.setTabStop(false);
labelS.setText("Start");

labelE.setLocation(new Point(128, 328));
labelE.setSize(new Point(32, 23));
labelE.setTabIndex(20);
labelE.setTabStop(false);
labelE.setText("End");

labelLS.setLocation(new Point(224, 328));
labelLS.setSize(new Point(100, 23));
labelLS.setTabIndex(21);
labelLS.setTabStop(false);
labelLS.setText("Large Item Set:");

labelStart.setLocation(new Point(80, 352));
labelStart.setSize(new Point(32, 23));
labelStart.setTabIndex(22);
labelStart.setTabStop(false);
labelStart.setText("");

labelEnd.setLocation(new Point(128, 352));
labelEnd.setSize(new Point(32, 23));
labelEnd.setTabIndex(23);
labelEnd.setTabStop(false);
labelEnd.setText("");

label_.setLocation(new Point(112, 352));
label_.setSize(new Point(8, 23));
label_.setTabIndex(24);
label_.setTabStop(false);
label_.setText("---");

labelLargeItemSet.setLocation(new Point(336, 328));
labelLargeItemSet.setSize(new Point(288, 23));
labelLargeItemSet.setTabIndex(26);
labelLargeItemSet.setTabStop(false);
```

```
    labelLargeItemSet.setText("");

    this.setNewControls(new Control[] {
                labelLargeItemSet,
                label_,
                labelEnd,
                labelStart,
                labelLS,
                labelE,
                labelS,
                labelTI,
                labelIAR,
                labelIN,
                labelItemNumber,
                labelD,
                labelDate,
                labelTID,
                labelTransactionID,
                labelNumber,
                labelMonitor,
                buttonRun,
                comboBoxMinimumIntervalLength,
                labelMinimumIntervalLength,
                labelMinimumSupport,
                comboBoxMinimumSupport,
                buttonAllocationDataFile,
                editTargetDataFileName,
                labelTargetDataset,
                labelAnancement});
  }

  /**
   * The main entry point for the application.
   *
   * @param args Array of parameters passed to the application
   * via the command line.
   */
  public static void main(String args[])
  {
    Application.run(new Form1());
  }
}
```

Class ILSList.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import com.ms.wfc.io.*;
import com.ms.wfc.util.*;

public class ILSList extends List
{
  public ILSList()
  {
    this.capInc = 10;
  }

  public void dispose()
  {
  }

  private long TimeDomainStartPoint;
  private long TimeDomainEndPoint;

  public void setTimeDomainStartPoint(long tDSP)
  {
    TimeDomainStartPoint = tDSP;
  }

  public void setTimeDomainEndPoint(long tDEP)
  {
    TimeDomainEndPoint = tDEP;
  }

  public long getTimeDomainStartPoint()
  {
    return TimeDomainStartPoint;
  }

  public long getTimeDomainEndPoint()
  {
    return TimeDomainEndPoint;
  }

  public void enqueueIntervalList(IntervalList iL)
  {
    this.enqueueItem(iL);
  }

  public IntervalList getIntervalList(int n)
  {
    return (IntervalList) this.getItem(n);
  }

  public int getAmountOfIntervalList()
  {
    return this.getSize();
  }
}
```

Class IntervalList.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import com.ms.wfc.io.*;
import com.ms.wfc.util.*;

public class IntervalList extends List
{
  public IntervalList()
  {
    this.capInc = 10;
  }

  public void dispose()
  {
  }

  private long IntervalLength;

  public void setIntervalLength(long iL)
  {
    IntervalLength = iL;
  }

  public long getIntervalLength()
  {
    return IntervalLength;
  }

  public void enqueueIntervalNode(IntervalNode iN)
  {
    this.enqueueItem(iN);
  }

  public void removeIntervalNode(int n)
  {
    this.removeItem(n);
  }

  public IntervalNode getIntervalNode(int n)
  {
    return (IntervalNode) this.getItem(n);
  }

  public int getAmountOfIntervalNode()
  {
    return this.getSize();
  }
}
```

Class IntervalNode.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import com.ms.wfc.io.*;
import com.ms.wfc.util.*;

public class IntervalNode extends List
{
  public IntervalNode()
  {
    this.capInc = 10;
  }

  public void diapose()
  {
  }

  private long StartTimePoint;
  private long EndTimePoint;

  public void setStartTimePoint(long sTP)
  {
    StartTimePoint = sTP;
  }

  public void setEndTimePoint(long eTP)
  {
    EndTimePoint = eTP;
  }

  public long getStartTimePoint()
  {
    return StartTimePoint;
  }

  public long getEndTimePoint()
  {
    return EndTimePoint;
  }

  public void enqueueLargeItemset(LargeItemset ll)
  {
    this.enqueueItem(ll);
  }

  public void insertLargeItemset(int n, LargeItemset ll)
  {
    this.insertItem(n, ll);
  }

  public void removeLargeItemset(int n)
  {
    this.removeItem(n);
  }

  public LargeItemset getLargeItemset(int n)
  {
    return (LargeItemset) this.getItem(n);
  }

  public int getAmountOfLargeItemset()
  {
    return this.getSize();
  }
```

```
  public boolean getEmptyLargeItemsetList()
  {
    return this.getEmpty();
  }

  public void trimLargeItemsetListToSize()
  {
    this.trimToSize();
  }
}
```

Class LargeItemset.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import com.ms.wfc.io.*;
import com.ms.wfc.util.*;

public class LargeItemset extends List
{
  public LargeItemset()
  {
    this.capInc = 10;
  }

  public void dispose()
  {
  }

  private int LargeItemsetSize;
  private long Count = 0;

  public void setLargeItemsetSize(int lIS)
  {
    LargeItemsetSize = lIS;
  }

  public int getLargeItemsetSize()
  {
    return LargeItemsetSize;
  }

  public void setCount(long c)
  {
    Count = c;
  }

  public long getCount()
  {
    return Count;
  }

  public void increaseCount()
  {
    Count ++;
  }

  public void enqueueLargeItem(LargeItem ll)
  {
    this.enqueueItem(ll);
  }

  public LargeItem getLargeItem(int n)
  {
    return (LargeItem) this.getItem(n);
  }

  public int getAmountOfLargeItem()
  {
    return this.getSize();
  }
}
```

Class LargeItem.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import com.ms.wfc.io.*;

public class LargeItem extends Object
{
  public LargeItem()
  {
  }

  public void dispose()
  {
  }

  private int LargeItemNumber;

  public void setLargeItemNumber(int lIN)
  {
    LargeItemNumber = lIN;
  }

  public int getLargeItemNumber()
  {
    return LargeItemNumber;
  }
}
```

Class Item.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import com.ms.wfc.io.*;

public class Item
{
  public Item()
  {
  }

  public void dispose()
  {
  }

  public int itemNumber;
}
```

Appendix B: Program TI-tree

Class TI-tree.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import com.ms.wfc.io.*;

public class TI-tree extends Form
{
  public TI-tree()
  {
    super();
    initForm();
    tree = new Tree();
  }

  public void dispose()
  {
    super.dispose();
    components.dispose();
  }

  String temporalDatabaseFileName = null;
  Tree tree;
  float minimumSupport;
  int minimumIntervalLength;
  Time createPatternBaseStartTime;
  Time createPatternBaseEndTime;
  Time mineLongestIntervalLargeItemsetStartTime;
  Time mineLongestIntervalLargeItemsetEndTime;

  private void menuItem3_click(Object source, Event e)
  {
    openFileDialog1.setDefaultExt("tdf");
    openFileDialog1.setFilter("Temporal Database File (*.tdf)|*.tdf");
    openFileDialog1.setFilterIndex(1);
    openFileDialog1.setInitialDir("C:\\Documents and Settings\\Documents");
    int dlgResult = openFileDialog1.showDialog();
    if (dlgResult == DialogResult.OK)
    {
      temporalDatabaseFileName = openFileDialog1.getFileName();
      this.setText("3-Levels : " + temporalDatabaseFileName);
    }
  }

  private void menuItem9_click(Object source, Event e)
  {
    long cPBT;
    createPatternBaseStartTime = new Time();
    tree.generatePatternBase(temporalDatabaseFileName);
    createPatternBaseEndTime = new Time();
    cPBT = createPatternBaseEndTime.toLong() - createPatternBaseStartTime.toLong();
    float cPBTC = (float) cPBT / (float) 10000000;
    label3.setText(String.valueOf(cPBTC));
  }

  private void menuItem8_click(Object source, Event e)
  {
    long mLILIT;
    mineLongestIntervalLargeItemsetStartTime = new Time();
    tree.mineIntervalLargeItemset(minimumSupport, minimumIntervalLength);
    mineLongestIntervalLargeItemsetEndTime = new Time();
    mLILIT = mineLongestIntervalLargeItemsetEndTime.toLong() -
mineLongestIntervalLargeItemsetStartTime.toLong();
```

```
  float mLILITC = (float) mLILIT / (float) 10000000;
  label4.setText(String.valueOf(mLILITC));
  MessageBox.show("Done");
}

private void comboBox1_selectedIndexChanged(Object source, Event e)
{
  if (comboBox1.getSelectedIndex() != -1)
  {
    minimumSupport = java.lang.Integer.parseInt(comboBox1.getSelectedItem().toString());
    minimumSupport = minimumSupport / 100;
  }
}

private void comboBox2_selectedIndexChanged(Object source, Event e)
{
  if (comboBox2.getSelectedIndex() != -1)
  {
    minimumIntervalLength = java.lang.Integer.parseInt(comboBox2.getSelectedItem().toString());
  }
}

Container components = new Container();
MainMenu mainMenu1 = new MainMenu();
MenuItem menuItem1 = new MenuItem();
MenuItem menuItem2 = new MenuItem();
MenuItem menuItem3 = new MenuItem();
OpenFileDialog openFileDialog1 = new OpenFileDialog();
MenuItem menuItem4 = new MenuItem();
MenuItem menuItem5 = new MenuItem();
MenuItem menuItem6 = new MenuItem();
MenuItem menuItem7 = new MenuItem();
MenuItem menuItem8 = new MenuItem();
MenuItem menuItem9 = new MenuItem();
Label label1 = new Label();
Label label2 = new Label();
Label label3 = new Label();
Label label4 = new Label();
Label label5 = new Label();
Label label6 = new Label();
ComboBox comboBox1 = new ComboBox();
ComboBox comboBox2 = new ComboBox();

private void initForm()
{
  menuItem3.setText("Temporal Database File .tdf");
  menuItem3.addOnClick(new EventHandler(this.menuItem3_click));

  menuItem2.setMenuItems(new MenuItem[] {menuItem3});
  menuItem2.setText("Open");

  menuItem1.setMenuItems(new MenuItem[] {menuItem2});
  menuItem1.setText("File");

  /* @designTimeOnly openFileDialog1.setLocation(new Point(904, 32)); */

  menuItem8.setText("All Interval Large Itemsets");
  menuItem8.addOnClick(new EventHandler(this.menuItem8_click));

  menuItem7.setMenuItems(new MenuItem[] {menuItem8});
  menuItem7.setText("Longest Interval");

  menuItem6.setMenuItems(new MenuItem[] {menuItem7});
  menuItem6.setText("Mine");

  menuItem9.setText("Using Temporal Database File");
  menuItem9.addOnClick(new EventHandler(this.menuItem9_click));

  menuItem5.setMenuItems(new MenuItem[] {menuItem9});
```

```
    menuItem5.setText("Pattern Base");

    menuItem4.setMenuItems(new MenuItem[] {menuItem5});
    menuItem4.setText("Create");

    mainMenu1.setMenuItems(new MenuItem[] {menuItem1, menuItem4, menuItem6});
    /* @designTimeOnly mainMenu1.setLocation(new Point(928, 8)); */

    this.setText("TI-tree");
    this.setAutoScaleBaseSize(new Point(5, 13));
    this.setClientSize(new Point(1016, 713));
    this.setMenu(mainMenu1);

    label1.setLocation(new Point(8, 656));
    label1.setSize(new Point(176, 23));
    label1.setTabIndex(0);
    label1.setTabStop(false);
    label1.setText("Creating Pattern Base Used (Sec.) :");

    label2.setLocation(new Point(8, 680));
    label2.setSize(new Point(208, 23));
    label2.setTabIndex(1);
    label2.setTabStop(false);
    label2.setText("Mining Interval Large Itemset Used (Sec.) :");

    label3.setLocation(new Point(184, 656));
    label3.setSize(new Point(208, 23));
    label3.setTabIndex(3);
    label3.setTabStop(false);
    label3.setText("");

    label4.setLocation(new Point(216, 680));
    label4.setSize(new Point(192, 23));
    label4.setTabIndex(2);
    label4.setTabStop(false);
    label4.setText("");

    label5.setLocation(new Point(696, 104));
    label5.setSize(new Point(128, 23));
    label5.setTabIndex(4);
    label5.setTabStop(false);
    label5.setText("Minimum Support % : ");

    label6.setLocation(new Point(696, 128));
    label6.setSize(new Point(128, 23));
    label6.setTabIndex(5);
    label6.setTabStop(false);
    label6.setText("Minimum Interval Length : ");

    comboBox1.setLocation(new Point(832, 104));
    comboBox1.setSize(new Point(121, 21));
    comboBox1.setTabIndex(9);
    comboBox1.setText("comboBox1");
    comboBox1.setItems(new Object[] {"5", "10", "15", "20", "25", "30", "35", "40", "45", "50","55",
"60", "65", "70", "75", "80", "85", "90", "95"});
    comboBox1.addOnSelectedIndexChanged(new
EventHandler(this.comboBox1_selectedIndexChanged));

    comboBox2.setLocation(new Point(832, 128));
    comboBox2.setSize(new Point(121, 21));
    comboBox2.setTabIndex(8);
    comboBox2.setText("comboBox2");
    comboBox2.setItems(new Object[] {"1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13",
"14", "15", "16", "17", "18", "19", "20", "21", "22", "23", "24", "25", "26", "27", "28", "29", "30"});
    comboBox2.addOnSelectedIndexChanged(new
EventHandler(this.comboBox2_selectedIndexChanged));

    this.setNewControls(new Control[] {comboBox2, comboBox1, label6, label5, label4, label3, label2,
label1});
```

```
  }
  public static void main(String args[])
  {
    Application.run(new TI-tree());
  }
}
```

Class Tree.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import com.ms.wfc.io.*;
import com.ms.wfc.util.*;

public class Tree
{
  private Node root;
  private TimeDomainTransactionCounts timeDomainTransactionCounts;

  public Tree ()
  {
    timeDomainTransactionCounts = new TimeDomainTransactionCounts();
    root = new Node(null, null, null, null, null);
  }

  private void compareAndAddNode(Node cAANTN, Node cAANCN)
  {
  }

  private void addNodeIntoTree (Node aNITN, long aNITG)
  {
    long granule = aNITG;
    Node currentNode;
    Node thisNode = aNITN;
    if (root.getFirstChild() == null)
    {
      thisNode.increasePatternCountOn(granule);
      root.setFirstChild(thisNode);
      thisNode.setFather(root);
    }
    else
    {
      currentNode = root.getFirstChild();
      char relationship;
      relationship = currentNode.compareNode(thisNode);
      while ((relationship == 'D') || (relationship == 'Y') || (relationship == 'N'))
      {
        if (relationship == 'D')
        {
          if (currentNode.getFirstChild() != null)
          {
            currentNode = currentNode.getFirstChild();
            relationship = currentNode.compareNode(thisNode);
            continue;
          }
          else
            break;
        }
        else
        {
          if (relationship == 'Y')
          {
            if (currentNode.getYoungerBrother() != null)
            {
              currentNode = currentNode.getYoungerBrother();
              relationship = currentNode.compareNode(thisNode);
              continue;
            }
            else
              break;
          }
          else
          {
```

```
        if (currentNode.getYoungerBrother() != null)
        {
          currentNode = currentNode.getYoungerBrother();
          relationship = currentNode.compareNode(thisNode);
          continue;
        }
        else
          break;
      }
    }
  }
  switch (relationship)
  {
    case 'N':
    {
      MessageBox.show("Error No Relationship");
      break;
    }
    case 'Y':
    {
      thisNode.increasePatternCountOn(granule);
      thisNode.setFather(currentNode.getFather());
      thisNode.setElderBrother(currentNode);
      currentNode.setYoungerBrother(thisNode);
      break;
    }
    case 'E':
    {
      if (currentNode.getElderBrother() == null)
      {
        Node f = currentNode.getFather();
        thisNode.increasePatternCountOn(granule);
        thisNode.setFather(f);
        thisNode.setYoungerBrother(currentNode);
        f.setFirstChild(thisNode);
        currentNode.setElderBrother(thisNode);
      }
      else //(currentNode.getElderBrother() != null)
      {
        Node eB = currentNode.getElderBrother();
        thisNode.increasePatternCountOn(granule);
        thisNode.setFather(currentNode.getFather());
        thisNode.setElderBrother(eB);
        thisNode.setYoungerBrother(currentNode);
        eB.setYoungerBrother(thisNode);
        currentNode.setElderBrother(thisNode);
      }
      break;
    }
    case 'S':
    {
      currentNode.increasePatternCountOn(granule);
      break;
    }
    case 'A':
    {
      MessageBox.show("Error Ancestor");
      break;
    }
    case 'D':
    {
      thisNode.increasePatternCountOn(granule);
      thisNode.setFather(currentNode);
      currentNode.setFirstChild(thisNode);
      break;
    }
    case 'R':
    {
      MessageBox.show("Error Root");
```

```
            break;
          }
        }
      }
    }
  }

  private ItemsetList generateItemsets (ItemSet gIIS)
  {
    ItemSet baseItemset = gIIS;
    ItemsetList itemsetList = new ItemsetList();
    Integer number;
    int baseItemsetSize = baseItemset.getItemSetSize();
    for (int i = 0; i < baseItemsetSize; i ++)
    {
      number = (Integer) baseItemset.getItem(i);
      itemsetList.addInNewItemNumber(number.intValue());
    }
    return itemsetList;
  }

  private void putIntoTree (long pITG, ItemSet pITIS)
  {
    Node node;
    ItemsetList itemsetList = generateItemsets(pITIS);
    timeDomainTransactionCounts.increaseGranuleCountOn(pITG);
    int itemsetListSize = itemsetList.getSize();
    for (int i = 0; i < itemsetListSize; i ++)
    {
      node = new Node((ItemSet) itemsetList.getItem(i), null, null, null, null);
      addNodeIntoTree(node, pITG);
    }
  }

  public void generatePatternBase (String gPBFN)
  {
    String fileName = gPBFN;
    String nextLine = null;
    long granule;
    long transactionID;
    int itemNumber;
    ItemSet itemset;
    TextReader reader;
    boolean getNewTransaction;
    long tID;

    itemset = null;
    transactionID = -1;
    granule = -1;
    getNewTransaction = true;
    reader = new TextReader(fileName);

    nextLine = reader.readLine();
    while (nextLine != null)
    {
      tID = java.lang.Long.parseLong(nextLine.substring(0, 6).trim());
      if (transactionID != tID)
      {
        getNewTransaction = true;
      }
      if (getNewTransaction == true)
      {
        if (itemset != null)
        {
          putIntoTree(granule, itemset);
          itemset = null;
        }
        transactionID = tID;
        granule = java.lang.Long.parseLong(nextLine.substring(6, 10).trim());
        itemset = new ItemSet();
```

```
        getNewTransaction = false;
      }
      itemset.addItemNumber(java.lang.Integer.parseInt(nextLine.substring(10, 15).trim()));
      nextLine = reader.readLine();
    }
    if (itemset != null)
    {
      putIntoTree(granule, itemset);
      itemset = null;
    }
    reader.close();
  }

  private Interval mineLongestInterval (float mLIMS, int mLIMIL, Node mLIN)
  {
    float minimumSupport = mLIMS;
    int minimumIntervalLength = mLIMIL;
    Node currentNode = mLIN;
    Interval first = null;
    Interval last = null;
    Interval interval;

//Step 1
    long begin = timeDomainTransactionCounts.getFirstGranule();
    long end = timeDomainTransactionCounts.getLastGranule();
    long firstS1 = begin;
    long secondS1 = begin + minimumIntervalLength -1;
    if (secondS1 > end)
    {
      return null;
    }
    while (secondS1 <= end)
    {
      if ((currentNode.getIntervalPatternCountBetween(firstS1, secondS1) >=
(timeDomainTransactionCounts.getIntervalGranuleCountBetween(firstS1, secondS1) *
minimumSupport)) && (timeDomainTransactionCounts.getIntervalGranuleCountBetween(firstS1,
secondS1) > 0))
      {
        interval = new Interval();
        interval.start = firstS1;
        interval.end = secondS1;
        if (first == null)
        {
          first = interval;
          interval.isFirst = true;
          last = interval;
          firstS1 ++;
          secondS1 ++;
          continue;
        }
        else
        {
          last.next = interval;
          last = interval;
          firstS1 ++;
          secondS1 ++;
          continue;
        }
      }
      else
      {
        firstS1 ++;
        secondS1 ++;
      }
    }
//Step 2
    Interval firstIntervalPoint = null;
    Interval secondIntervalPoint = null;
    Interval passedIntervalPoint = null;
```

```
    firstIntervalPoint = first;
    if (firstIntervalPoint != null)
    {
      secondIntervalPoint = firstIntervalPoint.next;
      while (secondIntervalPoint != null)
      {
        if ((firstIntervalPoint.start + 1 == secondIntervalPoint.start) && (firstIntervalPoint.end + 1 ==
secondIntervalPoint.end))
        {
          if ((currentNode.getIntervalPatternCountBetween(firstIntervalPoint.start,
secondIntervalPoint.end) >=
(timeDomainTransactionCounts.getIntervalGranuleCountBetween(firstIntervalPoint.start,
secondIntervalPoint.end) * minimumSupport)) &&
(timeDomainTransactionCounts.getIntervalGranuleCountBetween(firstIntervalPoint.start,
secondIntervalPoint.end) > 0))
          {
            interval = new Interval();
            interval.start = firstIntervalPoint.start;
            interval.end = secondIntervalPoint.end;
            last.next = interval;
            last = interval;
            firstIntervalPoint.canBeDeleted = true;
            secondIntervalPoint.canBeDeleted = true;
          }
        }
        passedIntervalPoint = firstIntervalPoint;
        firstIntervalPoint = secondIntervalPoint;
        secondIntervalPoint = secondIntervalPoint.next;
        if (passedIntervalPoint.canBeDeleted == true)
        {
          if (passedIntervalPoint.isFirst == true)
          {
            firstIntervalPoint.isFirst = true;
          }
          first = firstIntervalPoint;
          passedIntervalPoint = null;
        }
      }
    }
    else
    {
    }
    return first;
  }

  private Interval mineLongestInterval (float mLIMS, int mLIMIL, Node mLIN, long mLISG, long
mLIEG)
  {
    float minimumSupport = mLIMS;
    int minimumIntervalLength = mLIMIL;
    Node currentNode = mLIN;
    Interval first = null;
    Interval last = null;
    Interval interval;

//Step 1
    long begin = mLISG;
    long end = mLIEG;
    long firstS1 = begin;
    long secondS1 = begin + minimumIntervalLength -1;
    if (secondS1 > end)
    {
      return null;
    }
    while (secondS1 <= end)
    {
      if ((currentNode.getIntervalPatternCountBetween(firstS1, secondS1) >=
(timeDomainTransactionCounts.getIntervalGranuleCountBetween(firstS1, secondS1) *
```

```
minimumSupport)) && (timeDomainTransactionCounts.getIntervalGranuleCountBetween(firstS1,
secondS1) > 0))
      {
        interval = new Interval();
        interval.start = firstS1;
        interval.end = secondS1;
        if (first == null)
        {
          first = interval;
          interval.isFirst = true;
          last = interval;
          firstS1 ++;
          secondS1 ++;
          continue;
        }
        else
        {
          last.next = interval;
          last = interval;
          firstS1 ++;
          secondS1 ++;
          continue;
        }
      }
      else
      {
        firstS1 ++;
        secondS1 ++;
      }
    }
//Step 2
    Interval firstIntervalPoint = null;
    Interval secondIntervalPoint = null;
    Interval passedIntervalPoint = null;
    firstIntervalPoint = first;
    if (firstIntervalPoint != null)
    {
      secondIntervalPoint = firstIntervalPoint.next;
      while (secondIntervalPoint != null)
      {
        if ((firstIntervalPoint.start + 1 == secondIntervalPoint.start) && (firstIntervalPoint.end + 1 ==
secondIntervalPoint.end))
        {
          if ((currentNode.getIntervalPatternCountBetween(firstIntervalPoint.start,
secondIntervalPoint.end) >=
(timeDomainTransactionCounts.getIntervalGranuleCountBetween(firstIntervalPoint.start,
secondIntervalPoint.end) * minimumSupport)) &&
(timeDomainTransactionCounts.getIntervalGranuleCountBetween(firstIntervalPoint.start,
secondIntervalPoint.end) > 0))
          {
            interval = new Interval();
            interval.start = firstIntervalPoint.start;
            interval.end = secondIntervalPoint.end;
            last.next = interval;
            last = interval;
            firstIntervalPoint.canBeDeleted = true;
            secondIntervalPoint.canBeDeleted = true;
          }
        }
        passedIntervalPoint = firstIntervalPoint;
        firstIntervalPoint = secondIntervalPoint;
        secondIntervalPoint = secondIntervalPoint.next;
        if (passedIntervalPoint.canBeDeleted == true)
        {
          if (passedIntervalPoint.isFirst == true)
          {
            firstIntervalPoint.isFirst = true;
          }
          first = firstIntervalPoint;
```

```
            passedIntervalPoint = null;
          }
        }
      }
      else
      {
      }
      return first;
    }

    private void throughoutTree (float tTMS, int tTMIL, Node tTN)
    {
      mineLongestInterval(tTMS, tTMIL, tTN);
      if (tTN.getFirstChild() != null)
        throughoutTree(tTMS, tTMIL, tTN.getFirstChild());
      if (tTN.getYoungerBrother() != null)
        throughoutTree(tTMS, tTMIL, tTN.getYoungerBrother());
    }

    private void throughoutTree (float tTMS, int tTMIL, Node tTN, long tTSG, long tTEG)
    {
      Interval interval = mineLongestInterval(tTMS, tTMIL, tTN, tTSG, tTEG);
      while ((interval != null) && (tTN.getFirstChild() != null))
      {
        throughoutTree(tTMS, tTMIL, tTN.getFirstChild(), interval.start, interval.end);
        interval = interval.next;
      }
      if (tTN.getYoungerBrother() != null)
        throughoutTree(tTMS, tTMIL, tTN.getYoungerBrother(), tTSG, tTEG);
    }

    public void mineIntervalLargeItemset (float mILIMS, int mILIMIL)
    {
      if (root.getFirstChild() == null)
        return;
      else
        throughoutTree(mILIMS, mILIMIL, root.getFirstChild(),
  timeDomainTransactionCounts.getFirstGranule(), timeDomainTransactionCounts.getLastGranule());
    }

    public void mineIntervalLargeItemset (float mILIMS, int mILIMIL, long mILISG, long mILIEG)
    {
    }

    public void savePatternBase ()
    {
    }

    public void loadPatternBase ()
    {
    }

    private void showItemset (Node node)
    {
      String message = null;
      ItemSet itemset = node.getItemSet();
      int size = itemset.getItemSetSize();
      message = String.valueOf(size) + " :";
      int number;
      for (int i = 0; i < size; i ++)
      {
        number = itemset.getItemNumber(i);
        message += " " + String.valueOf(number);
      }
      MessageBox.show(message);
      if (node.getFirstChild() != null)
        showItemset(node.getFirstChild());
      if (node.getYoungerBrother() != null)
        showItemset(node.getYoungerBrother());
```

```
  }

  public void showPatternBase ()
  {
    Node node = root.getFirstChild();
    showItemset(node);
  }
}
```

Class Node.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;

public class Node
{
  private ItemSet itemSet = null;
  private Node father = null; //the reason of using this reference is to indicate whether this is a 1-
itemset (root)
  private Node youngerBrother = null;
  private Node firstChild = null;
  private Node elderBrother = null;
  private TimeSeriesPatternCounts timeSeriesPatternCounts;

  public Node (ItemSet nIS, Node nF, Node nYB, Node nFC, Node nEB)
  {
    itemSet = nIS;
    father = nF;
    youngerBrother = nYB;
    firstChild = nFC;

    timeSeriesPatternCounts = new TimeSeriesPatternCounts ();
  }

  public Node getFather ()
  {
    return father;
  }

  public Node getYoungerBrother ()
  {
    return youngerBrother;
  }

  public Node getElderBrother ()
  {
    return elderBrother;
  }

  public Node getFirstChild ()
  {
    return firstChild;
  }

  public ItemSet getItemSet ()
  {
    return itemSet;
  }

  public long getIntervalPatternCountBetween (long gPCOG)
  {
    return timeSeriesPatternCounts.getIntervalGranuleCountBetween(gPCOG);
  }

  public long getIntervalPatternCountBetween (long gIPCBS, long gIPCBE)
  {
    return timeSeriesPatternCounts.getIntervalGranuleCountBetween(gIPCBS, gIPCBE);
  }

  public void setFather (Node sFF)
  {
    father = sFF;
  }

  public void setYoungerBrother (Node sYBYB)
```

```
    {
      youngerBrother = sYBYB;
    }

    public void setElderBrother (Node sEBEB)
    {
      elderBrother = sEBEB;
    }

    public void setFirstChild (Node sFCFC)
    {
      firstChild = sFCFC;
    }

    public void increasePatternCountOn (long iPCOG)
    {
      timeSeriesPatternCounts.increaseGranuleCountOn(iPCOG);
    }

    public char compareNode(Node cNN)
    {
      return itemSet.compareItemSet(cNN.getItemSet());
    }
}
```

Class ItemSet.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import com.ms.wfc.io.*;
import com.ms.wfc.util.*;

public class ItemSet extends List
{
  private Integer itemNumber;

  public ItemSet ()
  {
    this.capInc = 1;
  }

  public void addItemNumber (int aINN)
  {
    itemNumber = new Integer(aINN);
    int thisSize = this.getSize();
    if (this.getEmpty() == true)
    {
      this.enqueueItem(itemNumber);
      return;
    }
    Integer testItemNumber;
    for (int i = 0; i < thisSize; i++)
    {
      testItemNumber = (Integer) this.getItem(i);
      if ((testItemNumber.intValue() < itemNumber.intValue()) && (i < thisSize - 1))
      continue;
      else
      {
        if (testItemNumber.intValue() == itemNumber.intValue())
          break;
        else
        {
          if (testItemNumber.intValue() > itemNumber.intValue())
          {
            this.insertItem(i, itemNumber);
            break;
          }
          else
          {
            if (i == thisSize -1)
            {
              this.enqueueItem(itemNumber);
              break;
            }
          }
        }
      }
    }
  }

  public char compareItemSet (ItemSet cISIS)
  {
    char noRelationship = 'N';
    char isYoungerBrother = 'Y';
    char isElderBrother = 'E';
    char isSame = 'S';
    char isAncestor = 'A';
    char isDescendant = 'D';
    char isRoot = 'R';

    int thisSize = this.getSize();
```

```
if (cISIS.getSize() == 0)//Empty itemset means root
  return isRoot;
else
{
  if (thisSize == cISIS.getSize())//same itemset size compare
  {
    if (thisSize == 1)//1-itemset special compare
    {
      Integer testThis = (Integer) this.getItem(0);
      Integer testCISIS = (Integer) cISIS.getItem(0);
      if (testThis.intValue() == testCISIS.intValue())
        return isSame;
      else
      {
        if (testThis.intValue() < testCISIS.intValue())
          return isYoungerBrother;
        else
          return isElderBrother;
      }
    }
    else//2 or more itemset
    {
      Integer tT;
      Integer tC;
      boolean ts1 = true;
      for (int i = 0; i < thisSize - 1; i++)
      {
        tT = (Integer) this.getItem(i);
        tC = (Integer) cISIS.getItem(i);
        if (tT.intValue() != tC.intValue())
        {
          ts1 = false;
          break;
        }
      }
      if (ts1 == true)
      {
        tT = (Integer) this.getItem(thisSize - 1);
        tC = (Integer) cISIS.getItem(thisSize - 1);
        if (tT.intValue() == tC.intValue())
          return isSame;
        else
        {
          if (tT.intValue() < tC.intValue())
            return isYoungerBrother;
          else
            return isElderBrother;
        }
      }
      else
        return noRelationship;
    }
  }
  else//different itemset size compare
  {
    if (thisSize < cISIS.getSize())
    {
      boolean ts2 = true;
      Integer tT2;
      Integer tC2;
      for (int i = 0; i < thisSize; i++)
      {
        tT2 = (Integer) this.getItem(i);
        tC2 = (Integer) cISIS.getItem(i);
        if (tT2.intValue() != tC2.intValue())
        {
          ts2 = false;
          break;
```

```
            }
          }
          if (ts2 == true)
            return isDescendant;
          else
            return noRelationship;
        }
        else
        {
          boolean ts3 = true;
          Integer tT3;
          Integer tC3;
          for (int i = 0; i < cISIS.getSize(); i++)
          {
            tT3 = (Integer) this.getItem(i);
            tC3 = (Integer) cISIS.getItem(i);
            if (tT3.intValue() != tC3.intValue())
            {
              ts3 = false;
              break;
            }
          }
          if (ts3 == true)
            return isAncestor;
          else
            return noRelationship;
        }
      }
    }
  }

  public int getItemSetSize ()
  {
    return this.getSize();
  }

  public int getItemNumber (int gINI)
  {
    int index = gINI;
    Integer numberObject = (Integer) this.getItem(index);
    return numberObject.intValue();
  }

  public ItemSet coloneItemset ()
  {
    ItemSet newItemset = new ItemSet();
    int thisSize = this.getSize();
    Integer copyNumber;
    for (int i = 0; i < thisSize; i ++)
    {
      copyNumber = (Integer) this.getItem(i);
      newItemset.addItemNumber(copyNumber.intValue());
    }
    return newItemset;
  }
}
```

Class ItemsetList.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import com.ms.wfc.io.*;
import com.ms.wfc.util.*;

public class ItemsetList extends List
{
  public ItemsetList ()
  {
    this.capInc = 10;
  }

  public void addInNewItemNumber (int aINININ)
  {
    int newItemNumber = aINININ;
    ItemSet itemset;
    ItemSet originalItemset;
    if (this.getSize() == 0)
    {
      itemset = new ItemSet();
      itemset.addItemNumber(newItemNumber);
      this.enqueueItem(itemset);
      return;
    }
    else
    {
      int currentItemsetListSize = this.getSize();
      for (int i = 0; i < currentItemsetListSize; i ++)
      {
        originalItemset = (ItemSet) this.getItem(i);
        itemset = (ItemSet) originalItemset.coloneItemset();
        itemset.addItemNumber(newItemNumber);
        this.enqueueItem(itemset);
      }
      itemset = new ItemSet();
      itemset.addItemNumber(newItemNumber);
      this.enqueueItem(itemset);
      return;
    }
  }
}
```

Class Interval.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import com.ms.wfc.io.*;
import com.ms.wfc.util.*;

public class Interval
{
  public long start;
  public long end;
  public Interval next;
  public boolean isFirst = false;
  public boolean canBeDeleted = false;
}
Class GranuleCount.java
public class GranuleCount
{
  private long granule;
  private long count;

  public GranuleCount (long gCG)
  {
    granule = gCG;
    count = 0;
  }

  public void increaseGranuleCount ()
  {
    count++;
  }

  public long getGranule ()
  {
    return granule;
  }

  public long getCount ()
  {
    return count;
  }
}
```

Class TimeDomainTransactionCounts.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import com.ms.wfc.io.*;
import com.ms.wfc.util.*;

public class TimeDomainTransactionCounts extends List
{
  private GranuleCount granuleCount;
  public TimeDomainTransactionCounts ()
  {
    this.capInc = 1;
  }

  public void increaseGranuleCountOn (long iGCOG)
  {
    if (this.getEmpty() == true)
    {
      granuleCount = new GranuleCount(iGCOG);
      granuleCount.increaseGranuleCount();
      this.enqueueItem(granuleCount);
      return;
    }
    int thisSize = this.getSize();
    for (int i = 0; i < thisSize; i++)
    {
      granuleCount = (GranuleCount) this.getItem(i);
      if ((granuleCount.getGranule() < iGCOG) && (i < thisSize -1))
        continue;
      else
      {
        if (granuleCount.getGranule() == iGCOG)
        {
          granuleCount.increaseGranuleCount();
          break;
        }
        else
        {
          if (granuleCount.getGranule() > iGCOG)
          {
            granuleCount = new GranuleCount(iGCOG);
            granuleCount.increaseGranuleCount();
            this.insertItem(i, granuleCount);
            break;
          }
          else
            if (granuleCount.getGranule() < iGCOG)
            {
              granuleCount = new GranuleCount(iGCOG);
              granuleCount.increaseGranuleCount();
              this.enqueueItem(granuleCount);
              break;
            }
        }
      }
    }
  }

  public long getIntervalGranuleCountBetween (long gGCOG)
  {
    long count = 0;
    if (this.getEmpty() == true)
    {
      return 0;
    }
```

```
    else
    {
      int thisSize = this.getSize();
      for (int i = 0; i < thisSize; i++)
      {
        granuleCount = (GranuleCount) this.getItem(i);
        if (granuleCount.getGranule() < gGCOG)
          continue;
        else
        {
          if (granuleCount.getGranule() == gGCOG)
            count = granuleCount.getCount();
          else
          {
            if (granuleCount.getGranule() > gGCOG)
              break;
          }
        }
      }
      return count;
    }
  }

  public long getIntervalGranuleCountBetween(long gIGCBS, long gIGCBE)
  {
    long count = 0;
    if (this.getEmpty() == true)
    {
      return 0;
    }
    else
    {
      int thisSize = this.getSize();
      for (int i = 0; i < thisSize; i++)
      {
        granuleCount = (GranuleCount) this.getItem(i);
        if (granuleCount.getGranule() < gIGCBS)
          continue;
        else
        {
          if (granuleCount.getGranule() > gIGCBE)
            break;
          else
            count += granuleCount.getCount();
        }
      }
      return count;
    }
  }

  public long getFirstGranule ()
  {
    if (this.getSize() > 0)
    {
      granuleCount = (GranuleCount) this.getItem(0);
      return granuleCount.getGranule();
    }
    else
      return -1;
  }

  public long getLastGranule ()
  {
    if (this.getSize() > 0)
    {
      granuleCount = (GranuleCount) this.getItem(this.getSize() - 1);
      return granuleCount.getGranule();
    }
    else
```

```
        return -1;
    }
}
```

Class TimeSeriesPatternCounts.java

```
import com.ms.wfc.app.*;
import com.ms.wfc.core.*;
import com.ms.wfc.ui.*;
import com.ms.wfc.html.*;
import com.ms.wfc.io.*;
import com.ms.wfc.util.*;

public class TimeSeriesPatternCounts extends List
{
  private GranuleCount granuleCount;
  public TimeSeriesPatternCounts ()
  {
    this.capInc = 1;
  }

  public void increaseGranuleCountOn (long iGCOG)
  {
    if (this.getEmpty() == true)
    {
      granuleCount = new GranuleCount(iGCOG);
      granuleCount.increaseGranuleCount();
      this.enqueueItem(granuleCount);
      return;
    }
    int thisSize = this.getSize();
    for (int i = 0; i < thisSize; i++)
    {
      granuleCount = (GranuleCount) this.getItem(i);
      if ((granuleCount.getGranule() < iGCOG) && (i < thisSize -1))
        continue;
      else
      {
        if (granuleCount.getGranule() == iGCOG)
        {
          granuleCount.increaseGranuleCount();
          break;
        }
        else
        {
          if (granuleCount.getGranule() > iGCOG)
          {
            granuleCount = new GranuleCount(iGCOG);
            granuleCount.increaseGranuleCount();
            this.insertItem(i, granuleCount);
            break;
          }
          else
            if (granuleCount.getGranule() < iGCOG)
            {
              granuleCount = new GranuleCount(iGCOG);
              granuleCount.increaseGranuleCount();
              this.enqueueItem(granuleCount);
              break;
            }
        }
      }
    }
  }

  public long getIntervalGranuleCountBetween (long gGCOG)
  {
    long count = 0;
    if (this.getEmpty() == true)
    {
      return 0;
    }
```

```
    else
    {
      int thisSize = this.getSize();
      for (int i = 0; i < thisSize; i++)
      {
        granuleCount = (GranuleCount) this.getItem(i);
        if (granuleCount.getGranule() < gGCOG)
          continue;
        else
        {
          if (granuleCount.getGranule() == gGCOG)
            count = granuleCount.getCount();
          else
          {
            if (granuleCount.getGranule() > gGCOG)
              break;
          }
        }
      }
      return count;
    }
  }

  public long getIntervalGranuleCountBetween(long gIGCBS, long gIGCBE)
  {
    long count = 0;
    if (this.getEmpty() == true)
    {
      return 0;
    }
    else
    {
      int thisSize = this.getSize();
      for (int i = 0; i < thisSize; i++)
      {
        granuleCount = (GranuleCount) this.getItem(i);
        if (granuleCount.getGranule() < gIGCBS)
          continue;
        else
        {
          if (granuleCount.getGranule() > gIGCBE)
            break;
          else
            count += granuleCount.getCount();
        }
      }
      return count;
    }
  }
}
```