MINING AND ANALYSIS OF TIME-STAMPED DATABASES

A Thesis submitted to Goa University for the Award of the Degree of

DOCTOR OF PHILOSOPHY

in

Computer Science and Technology

By

Jhimli Adhikari

Department of Computer Science

Narayan Zantye College of Commerce

Bicholim, Goa 403 529

Research Guide

Dr. P. R. Rao, Professor

Department of Computer Science and Technology

Goa University,

Taleigao Goa

2012

Dedication

To my parents

Gouri Sankar Datta

and

Rina Datta

Acknowledgement

I would like to thank my advisor, Dr. P. R. Rao, for his support and advices during past five years.

I would like to thank University Grants Commission, India for sponsoring me on faculty improvement programme with leave to take up full time research.

I am grateful to Sri Arun Sakhardande, Principal, Narayan Zantye College of Commerce, Bicholim, Goa, for processing my study leave application at the right time.

I would like to thank to my colleagues for their supports during the period of my study leave.

I am also grateful to my husband, Animesh, my brother, Supriyo and my son, Sohom, for their moral supports.

Statement from student

As required under ordinance OB-9.9 of Goa University, I state that the present Ph D thesis entitled "Mining and Analysis of Time-stamped Databases" is my original contribution and the same has not been submitted on any previous occasion. To the best of my knowledge, the present study is the first comprehensive work of its kind in the area of Data Mining and Knowledge Discovery.

The literature related to the problem investigated has been cited. Due acknowledgements have been made whenever facilities and suggestions have been availed of.

Jhimli Adhikari

Department of Computer Science

Narayan Zantye College of Commerce

Bicholim, Goa 403 529

India

Date: 23/01/2012

Certificate

This is to certify that the Ph D thesis entitled "Mining and Analysis of Time-stamped Databases", submitted by Jhimli Adhikari for the award of degree of Doctor of Philosophy in Computer Science and Technology is based on her original work carried out under my supervision. The thesis or any part thereof has not been previously submitted for any other degree or diploma in any other University or Institute.

Place: Goa University

Dr. P. R. Rao, Professor

Department of Computer Science and Technology

Goa University, Taleigao Plateau

Goa-403206, India

List of publications

(2 journal papers, 1 book chapter and 2 conference papers)

1. *International Journal papers*

[1] Adhikari, J., Rao, P. R., Adhikari, A., "Clustering items in different data sources induced by stability", *International Arab Journal of Information Technology*, 6(4), 394–402, 2009 (impact factor 0.39 in 2012, cited by 8 papers)

[2] Adhikari, J., Rao, P. R., "Measuring influence of an item in a database over time", *Pattern Recognition Letters*, 31(3), 179–187, 2010 (impact factor 1.266 in 2012, cited by 1 paper)

2. *International Book chapter*

[1] Adhikari, J., Rao, P. R, Identifying calendar-based periodic patterns, S. Ramanna, L. Jain and R. J. Howlett (editors), *Emerging Paradigms in Machine Learning*, pp. 329–357, Springer, 2013

3. *International Conference paper*

[1] Adhikari, J., Rao, P.R., "Clustering items in multiple databases induced by stability", Proceedings of the *International Conference on Emerging Technologies and Applications in Engineering, Technology and Sciences*, pp. 370-375, 2008

[2] Adhikari, J., Rao, P. R., Pedrycz, W., "Mining icebergs in time-stamped databases", *Indian International Conference on Artificial Intelligence*, pp. 639-658, 2011

Synopsis

Data mining on time-stamped data deals with discovering various types of knowledge hidden in time-stamped data. Knowledge discovery in a time-stamped database is an interesting and well-known research issue (Adhikari & Rao, 2009; Mahanta et al, 2005; Mahanta et al., 2008). A vast amount of temporal data is available in science, engineering and medical fields. Also, many large companies collect data for a long period of time. Knowledge extracted from such data would help the companies to make better decisions. Due to the existence of large class of temporal datasets, applications dealing with temporal patterns seem to be present everywhere (Bettini et al, 2000; Hsu et al, 2007; Lattner, 2007; Mitsa, 2010). The goal of the thesis is to mine different time-stamped data and provide various types of data analyses.

Contributions made in this thesis are kept in Chapters 2, 3, 4, and 5. In Chapter 1, we discuss different concepts such as types of time-stamping (Mitsa, 2010), and time granularity (Bettini et al, 2000). We have made a comparison between time-stamped data and time series data (Brockwell, & Davis, 2002). Although there are various preprocessing techniques exist before mining data we illustrate only aggregation and partitioning, since these tasks are relevant to our study. We present various temporal patterns existing in the literature such as frequent patterns, temporal association rule, event, sequential patterns, episode, and temporal relational interval pattern. At the later part different data mining tasks viz., prediction, clustering, classification, search and

retrieval, pattern discovery have been illustrated. Finally, recent developments of temporal data mining in various fields have been surveyed.

Chapter 2 has been entitled as "*Clustering items in different data sources induced by stability*". Chapter 3 deals with the association among items and entitled as "*Measuring influence of an item in a database over time*". Chapter 4 is related to unusual pattern and entitled as "*Mining icebergs from time-stamped databases*". Finally, the Chapter 5 detects calendar based pattern and has been entitled as "*Identifying calendar-based periodic patterns*". In the following paragraphs, we describe the work performed in different chapters of the thesis.

The variation of sales of an item over time is an important issue. Many important decisions are based on items whose support variation is less over the time. These items are called as stable items. Stable items are useful in making many strategic decisions for a company. In Chapter 2 we have proposed a model of mining global patterns in multiple transactional time-stamped databases. Thus, we introduce the notion of stability of an item. The degree of stability is based on the variations of means and autocovariances. The proposed clustering technique is based on the notion of degree of stability of an item. The clustering technique requires computing the degree of variation for each item in the databases. Given a set of yearly databases, the difference in variations between every pair of items could be expressed by a square matrix, called *difference in variation*. This matrix is symmetric square matrix. Intuitively, if the difference in variations between two items is close to zero then they may be put in the same class. We have proposed the notion of best cluster by considering average degree of variation of a class. Also, we have designed

an alternative algorithm to find best cluster among items in different data sources. Experimental results are provided on two real and one synthetic transactional database.

In transactional database measuring influence among itemsets over time becomes an important issue, since many companies possess data for a long period of time so that they could be exploited in an efficient manner. Such analyses might be interesting since the proposed measure of influence considers both positive and negative influence of an itemset on another itemset. Tan et al. (2003) presented an overview of twenty one interestingness measures proposed in the statistics, machine learning, and data mining literature. Based on these observations, we consider five out of twenty one interestingness measures since overall influence of an itemset on another itemset lies in [-1, 1]. We show that none of the five measures serves as a measure of overall influence between two itemsets. Using the notion of overall influence, we have designed two algorithms for influence analysis involving specific items in a database. The first algorithm reports all the significant influences in a database. In the second algorithm we have sorted items based on their influences on a set of specific items. As the number of databases increases on a yearly basis, we have adopted incremental approach in these algorithms. Experimental results are reported for both synthetic and real-world databases.

It might be interesting as well as useful to know the interesting changes in sales over time. In Chapter 4 we have introduced a new pattern, called notch, of an item in time-stamped databases. First, we have introduced the notion of notch in a sales series of an item. Based on this pattern we have introduced two more patterns viz., generalized notch and iceberg notch, in sales series of an item. Iceberg notch represents a special sales

pattern of an item over time. It could be considered as an exceptional pattern in time-stamped databases. Study of such patterns could be important to understand the purchase behaviour of customers. It helps identifying the reasons of such behaviour. We have designed an algorithm for mining interesting icebergs in time-stamped databases. We have presented experimental results on four real databases and two synthetic databases.

A calendar-based periodic pattern is dependent on the schema of a calendar. We assume that the schema of the calendar-based pattern is based on day, month and year. In the recent time, researchers have reported an algorithm for finding calendar-based periodic pattern in a time-stamped data and introduced the concept of certainty factor in association with an overlapped interval (Mahanta et al., 2008). In Chapter 5 we mined locally frequent itemsets along with the set of intervals and their support range. We have extended the concept of certainty factor by incorporating support information for better analysis of overlapped intervals. Using this concept one can extract various calendar-based patterns viz., yearly, monthly, weekly and daily. In addition, we check whether any periodicity (full / partial) exists in the patterns. We have proposed some improvements in the algorithm for identifying calendar-based periodic pattern in a time-stamped dataset by introducing suitable data structure. The algorithm is incremental in nature. We have presented extensive data analysis on three data sets. We also analysed the constraints *mininterval*, *minsupp* and *maxga*p associated with each interval. We also provide a comparative analysis with our algorithm and the most recent algorithm for mining calendar-based periodic patterns. Experimental results are provided on real and synthetic dataset.

Table of Contents

Table of Contents
(continued)

Table of Contents
(continued)

Table of Contents
(continued)

Table of Contents
(continued)

Chapter 1

Preliminary Concepts

1.1 Introduction

Data mining and knowledge discovery process traditionally involves analysing a static dataset, where data instances are collected, stored and analysed to derive models, and eventually decisions are made based on these models (Han & Kamber, 2006). Many business processes are required to collect a huge volume of data over a long period of time. Data mining methods assume implicitly that the domain under consideration is stable over time, and thus provide a rather static view on knowledge hidden in the data available so far. But many real databases are dynamic, and hence grow over time. Moreover, many real world databases contain time information. For instance, data about the sold items including its production date, data about the sales includes selling date, data about the warranty case together with the date of the claim, the expiry date of warranty and the payment, etc. Time stamp seems to be a natural attribute to objects described in a database. Knowledge discovery in a time-stamped database is an interesting and well known research issue (Bettini et al., 1998; Mitsa, 2010; Hsu et al., 2008).

Data mining and knowledge discovery processes often involve analyzing data by ignoring time. Most of the cases temporal data are treated as an unordered collection of events. Analyzing data involving temporal dimension has received attention in the recent time (Adhikari & Rao, 2010; Leonard & Wolfe, 2005). Data analyses based on time dimension might offer significant knowledge to an organization. Time component is present almost in every data. Thus, time-based data analyses are present everywhere.

Temporal data mining (Antunes & Oliveira, 2001; Laxman & Sastry, 2006) is concerned with mining of a large sequential dataset ordered with respect to time. Time series data, time-stamped

data are examples of popular classes of sequential data, where data are ordered by time. One could consider a time-stamped database as a sequence of time databases.

In case of mining such sequence of multiple time-stamped databases an incremental approach (Adhikari & Rao, 2010; Lee et al., 2001) seems to be a natural and effective solution. Little work has been reported on mining and analysis of multiple time-stamped databases.

Many organizations transact data from multiple branches. Consider one such organization that possesses multiple databases. It might be difficult to mine all the databases together, since some of the branch (local) databases could be very large. In this situation a mono-database mining technique (Agrawal et al., 1994; Han et al., 2000) might fail. Local pattern analysis (Adhikari & Rao, 2008a; Zhang et al., 2003) is an important approach to mining multiple databases. In this approach each local database is mined locally. Then all the local pattern bases are analysed and / or synthesized for mining global patterns (Adhikari & Rao, 2008a; Wu & Zhang, 2003). In local pattern analysis we mine each database separately. This approach could also be applied to analysing a time-stamped database by considering it as a sequence of time databases. Web sites and transactional databases contain a large amount of time-stamped data related to an organization's suppliers and / or customers over time. Mining such time-stamped data could help business leaders make effective decisions by listening to their suppliers or customers via their transactions collected over time. Consider an established company possessing data over fifty consecutive years. Generally, the sales of a product vary from one season to another season. Also, a season re-appears on a yearly basis. Thus, one may decide dividing the entire database into a sequence of yearly databases. In this context, a yearly database could be considered as a

time database. Each of these time databases is similar to a branch database of a multi-branch company possessing multiple databases. Thus, mining and analysis of time databases is similar to local pattern analysis. The goal of this thesis is to mine and provide various types of data analyses based on time databases.

1.2 Time in databases

Temporal databases capture time-related attributes whose value may change with time. These databases contain time-stamping information. Time-stamping could be specified as follows (Mitsa, 2010):

- With a *valid time*, which is the time that the element information is true in the real world. For example, "The patient was admitted to the hospital on 5:15 a.m., March 3, 2005", (Tansel et al., 1993; Date et al., 2002).
- With a *transaction time*, which is the time the element information is entered into the database (occurrence time) (Tansel et al., 1993; Date et al., 2002).
- A table that contains both valid times and transaction times is said to be *bi-temporal*.

Many time-stamped databases are not related with valid time, since the records do not contain temporal information. On the other hand, almost all time-stamped databases are related with transaction time. In our work we have dealt with temporal databases containing transaction time. Market basket data are inherently related with time. Each record in market basket data is time-stamped at the time of checking out from the store. Time-stamped data could be transformed into events, time intervals, and time series.

Two types of temporal data are dominant in the development of temporal data mining. They are time-series data and sequence data. If the data contain numerical values, such as stock price,

rainfall, we usually say that the data is time series. On the other hand, when the data is based on categorical values, it is called sequence data. These can be time-stamped at regular or irregular time intervals. A common example is the items purchased by a customer in a supermarket. We have mainly worked with sequence data where the records in the database are entered according to their occurrence times.

Many problems on sequence data are based on the order of data points rather than their concrete time-stamps. In Chapters 2, 3, and 5, we have presented problems based on the order of data points. But the problem in Chapter 4 is based on both the order of the data points and their time-stamps.

1.3 Time granularity

Depending on the purpose or application different sizes of time unit may be appropriate. So the age of volcanoes may be measured in year, or decade, or hundred of years. The age of motorcars may be measured in year, or perhaps month for new cars. The age of babies may be measured in year, or month, or week, or day, and the age of bacteria in second or millisecond. The size of the unit in a particular scenario is referred to as the granularity of the unit; small temporal grains refer to short units of time (day, hour, second, millisecond, etc.), and large temporal grains refer to longer units of time (month, year, decade, etc.). Calendar units such as year, month and day; clock units such as hour, minute and second; and specialized units such as business day, holiday and academic year serve major roles in a wide range of information system applications.

Different facts may be associated with temporal contexts expressed in terms of different granularities. For example, a bank transaction may require a timestamp in seconds, while the presence of an employee in a department may be expressed in days. A temporal database that

allows facts expressed in terms of different granularities is called a temporal database with multiple granularities. Goralwalla et al. (2001) showed an approach to handling of multiple granularities in temporal data. Authors separated temporal data into two groups: anchored (calendrical day or month such as January 1st, 2008 or May 1978), and unanchored (time intervals such as 2 months, 5 h 20 min, etc.) data (Euzenat & Montanari, 2005). Thus, a temporal granule is a special kind of unanchored temporal data (Goralwalla et al., 1998). Cotofrei and Stoffel (2009) defines formalism for a specific temporal data mining task such as the discovery of rules inferred from database of events having a temporal dimension. The proposed theoretical framework, based on first-order temporal logic allows the definition of notions such as event, temporal rule, and confidence in a formal way. This formalism is then extended to include the notion of temporal granularity, and a detailed study is made to investigate the formal relationships between the support measures of the same event in linear time structures with different granularities.

1.4 Time-stamped data versus time series data

In businesses we often wish to discover knowledge from temporal databases. Both time stamped and time series data are related with time, where a time point is an instance of time with a given base granularity such as a second, minute, day, month, year, etc. We assume that the time points are always defined over a sequentially ordered domain of base values, and thus can be compared. A pair of time points defines a time interval. For example, a semi-closed time interval, denoted by $[t_1, t_2)$, is the finite set of base granularity time points t such that $t_1 \leq t < t_2$.

Time-stamped data is represented using observations at discrete points of time. For example, web-log, point-of-sale data, bank transactions, inventory, and stock market data. Transactional

data are time-stamped data collected over time at no particular frequency (Leonard & Wolfe, 2005). But time series data are collected over time at a particular frequency. Some examples of time series data are web-site visits per hour, sales per month, inventory draws per week, calls per day, and trades per weekday. The accumulation of time-stamped data into time series data is based on a particular frequency. The frequency associated with the time series varies with the problem at hand. Time-stamped data can be accumulated to form hourly, daily, weekly, monthly, or yearly time series. In addition, the method of accumulating data within each time period is based on a particular statistics. For example, sum, mean, median, minimum, maximum, standard deviation, and other statistics can be used to accumulate data within a particular time period.

A time series is a set of unique time points with values or objects assigned to each time point. But a time sequence is a multi-set of time points with assigned values or objects, i.e., it can include duplicate time points. If time stamps in the stored data are equidistant, data actually are time series. A time series can be univariate or multivariate. A multivariate time series is created by more than one variable while in a univariate time series there is one underlying variable. Another characteristic of time series is its stationarity (Brockwell & Davis, 2002). A stationary time series has a mean and a variance that does not change over time, while a non-stationary one has no salient mean and can decrease or increase over time.

1.5 Preprocessing of time-stamped data

Data preprocessing is an important step in the knowledge discovery process. Data must be appropriately preprocessed before deriving meaningful knowledge. Data preprocessing may vary from one application to another application. In the context of time-stamped data, there are two

important aspects of data preprocessing viz., aggregating the time-stamped data and partitioning the database at different levels of granularity.

1.5.1 Temporal aggregation

Temporal aggregation is a process in which a time line is partitioned over time and the values of various attributes in the database are accumulated over these partitions (Dumas et al., 1998). A typical example of temporal aggregation is the monthly accumulation of salary payment. Due to the large varieties of temporal data and their distribution over the time line, efficient algorithms to perform temporal grouping are necessary. Moon et al. (2003) proposed several methods for large-scale temporal aggregation. In this context, the choice of time granularity is an important issue as the characteristics of temporal patterns is heavily dependent on this parameter. Let us consider an online shop that acquires monthly reports from their web hosts. The web hosts deliver activity reports at regular intervals. Here time granularity refers to a month instead of a year. Therefore, for this application month-wise time-stamped data could be accumulated to form smaller databases. Similarly for stock market applications, weekly data accumulation may be preferred to monthly data. In Chapter 3 we have applied aggregation technique for finding sales of different items. One of the reasons to summarize the time-stamped data is to reduce the amount of data. Afterwards, an appropriate algorithm could be designed to handle reduced data.

1.5.2 Partitioning database into different levels of granularity

Consider an organization possessing data over fifty consecutive years. The organization might be interested in mining knowledge for various purposes. It may require partitioning a database for various purposes such as finding items whose supports are stable over the time (Adhikari et al., 2009), discovering the abrupt variation in sales of items (Adhikari et al., 2011), and extracting

yearly periodic patterns (Adhikari & Rao, 2011). In order to extract such knowledge, one could divide the given database into a number of yearly databases.

Also for the purpose of mining change (Bottcher et al., 2008) one may require partitioning a given database. In a prediction problem one requires analyzing past events that could originate from previous databases. Given such a problem, it might be strategically necessary to divide a large database into smaller databases. One could call these smaller databases as *time databases*. While dividing a large database one needs selecting certain time period. Selection of time period is an important decision and it is dependent on the problem. For many applications, such as the problems considered in Chapters 2, 3, 4 and 5, we divided the given database into yearly databases. We have considered time granularity as one year since a season re-appears on a yearly basis and the customers' purchase patterns might vary from season to season.

1.6 Temporal patterns

A *pattern* is a local structure that makes a specific statement about a few variables or data points. Spikes, for example, are patterns in a real-valued time series that may be of interest. The objective of pattern mining is simply to unearth all patterns of interest. Temporal data mining covers data analyses related to time. Matching and discovery of temporal patterns are very useful in many applications. There are many different approaches of temporal pattern mining based on various data models. They are usually designed with a particular application in mind. Several basic choices have to be made while mining temporal data. Usually both the concepts of *time point* (*instant*) and *time interval* have been used in the data mining literature to represent time (Terenziani & Snodgrass, 2004; Toman, 1996). These concepts are usually related to instantaneous events, or to situations lasting for a span of time. Care needs to be taken in

associating these concepts with the entities. For example, web login could be considered an instantaneous event, but observation on condition of a patient during ICU staying is an interval-based concept. Intervals are then represented by their upper and lower temporal bounds (start and end time points). Time intervals are often used to obtain representations of sets of time instants. In such a case, validity over a time period is interpreted as validity at every time instant belonging to it. In practice, most systems employed in real life applications have used a time point or time intervals. In the following sections we discuss a few interesting time point patterns as well as time interval patterns.

1.6.1 Frequent pattern

A frequent pattern is one that occurs many times in a database. Since the beginning of the 1990s, frequent pattern mining has become one of the most actively researched topics in data mining and knowledge discovery. Much of data mining literature is concerned with formulating useful pattern structures and developing efficient algorithms for discovering patterns that occur frequently in the data (Agrawal et al., 1993). The starting point was market basket analysis and especially the task of mining transactional data, which describes the shopping behavior of customers of supermarkets, online shops, for products that are frequently bought together. For this task, which became generally known as frequent itemset mining, a large number of efficient algorithms were developed, which are based on sophisticated data structures and clever processing schemes. Among them Apriori (Agrawal & Srikant, 1994), Eclat (Zaki, 2000b), and FP-growth (Han et al., 2000) are most widely known. The *support* (Agrawal et al., 1993) of an itemset is defined as the fraction of transactions containing the itemset. It has been used

extensively in identifying different types of patterns in a database including temporal data and as well as interval based data.

Periodic frequent patterns are special kind of frequent patterns that occur periodically (regularly) within a dataset (Tanbeer et al., 2009). In this approach, the time of occurrence of each transaction is taken into account for periodic frequent pattern mining.

1.6.2 Temporal association rule

Methods for finding frequent patterns are considered important because they can be used for discovering useful rules. These rules can in turn be used to infer some interesting regularities in the data. A rule consists of a pair of Boolean-valued propositions, a left-hand side proposition (the antecedent) and a right-hand side proposition (the consequent). The rule states that when the antecedent is true, then the consequent will be true as well. Rules have been popular representations of knowledge in machine learning and AI for many years. In data mining, *association rules* are used to capture association between different set of attributes in the data (Agrawal et al., 1993).

Temporal association rule can be defined as a pair (*R*, *T*), where *R* is an association rule and *T* is a temporal feature, such as a period or a calendar. There are three interesting measures regarding the discovery of association rules viz., support, confidence (Agrawal et al., 1993) and informativeness (Smyth & Goodman, 1992). The conditional probability of the consequent occurring given the antecedent is referred to as *confidence* of the rule. Informativeness is a measure that computes the usefulness of a rule in terms of the information it provides.

1.6.3 Event

Temporal events occur in a wide range of applications in business, government, and science.

Some of these events can be aggregated over time in a meaningful way and thus can be presented in time series visualizations. But other applications require each event to be visible. An event is a single, time-stamped item or a data point in time, which can be time-stamped. For a more systematic analysis, events are therefore briefly categorized as (i) event sequence, and (ii) event episode. An event sequence is a set of events that are ordered in time, whereas an event episode is a set of events that are time-stamped. Event sequences are investigated in order to predict events or to determine correlations of events (Agrawal et al., 1995). There is a distinction between event sequences and event episodes (Mannila et al., 1997). Since an event episode is a set of events that are time-stamped, the distance between the atomic events matters. Under the assumption that every event has an assigned value, event data can be further refined into a) time-synchronous event data, in which an accurate time-stamp is important, b) ordinal event data, where the ordering of the events according to time plays an important role, c) aggregateable event data, which can be summarized for a particular interval, and d) hierarchical event data, where the grouping is defined based on a hierarchical structure in the meta data. To foster a better understanding of analysis tasks for event data, we define the following terms. A *significant event* is a single event that is interesting for some reason. An *event cluster* is a set of events that are considered as being similar to each other. This may, but not necessarily, include similarity in time. An *event pattern* is an event sequence or episode that shows some interesting regularity with respect to certain properties.

1.6.4 Sequential pattern

Sequential pattern mining has received a particular attention in the last decade (Agrawal & Srikant, 1994; Zaki, 2001; Han, et al., 2000; Wang & Han, 2004). A sequence is a time-ordered

list of objects. These can be time-stamped at regular or irregular time intervals. The objective is to extract patterns from a set of sequences of instantaneous events that satisfy some user-specified constraints. These constraints can vary from just a support threshold that defines frequency of a set of gaps, windows (Zaki, 2000a; Srikant & Agrawal, 1996), or regular expression constraints (Garofalakis et al., 1999) in view of focusing more into the mining process.

An example of a temporal sequence is the time-stamped sequence of purchases of a customer on a web site. An event can be considered as a special case of a temporal sequence with a time-stamped element. Therefore, a series of events is another way to represent a temporal sequence, where the elements of the sequence are semantically of the same type. The sequential pattern mining framework is basically an extension of the idea of frequent itemsets having a temporal order. Here the database is not just some unordered collection of transactions. Each transaction in the database carries a time-stamp as well as a customer identifier. The transactions associated with a single customer can be regarded as a sequence of itemsets ordered by time and database contains one such transaction sequence corresponding to each customer. This concept of sequential patterns is quite general and can be used in many other situations as well. Sequential pattern mining can be used to discover those sequences of websites that are frequently visited one after another.

1.6.5 Episode

Another approach to discover temporal patterns in sequential data is the frequent episode discovery framework (Mannila et al., 1997). In the sequential patterns mining framework a collection of sequences are given, and the task is to discover ordered sequences of items that

occur in sufficiently many of those sequences. In the frequent episodes mining framework the data are given as a single long sequence, and the task is to unearth temporal patterns, called episode, and it occurs sufficiently often along that sequence.

An episode is defined as a sequence of events appearing in a specific order within a specific time window. An example of episode is the occurrence of flu followed by pneumonia. An event sequence is defined as a pair of events and corresponding timestamps $<(E_1, t_1), (E_2, t_2), \ldots >$, where t_i is an integer denoting the time stamp of the i-th event E_i. The sequence is ordered with respect to the timestamps so that $t_i \leq t_{i+1}$ for $i = 1, 2, \ldots$. An episode is just a partially ordered set of event types. An episode is said to belong to a sequence if the events in the episode appear in the same order in the sequence. An episode can be either serial or parallel. In case of serial episode order is important. An example of a serial episode is $(A \rightarrow B \rightarrow C)$, while a parallel episode is of the form (ABC). The problem of mining episodes is to discover all episodes that satisfy a minimum frequency threshold among the time windows with a user-specified window size. The frequent episode discovery framework has also been applied to many other kinds of datasets such as web navigation logs (Casas-Garriga, 2003) and Wal-Mart sales data (Atallah et al., 2004).

1.6.6 Temporal relational interval pattern

Time stamp could be expressed in either absolute or relative terms. Earlier we have discussed absolute value of time information, i.e. time point and time interval. The temporal relationship between two temporal events can be captured by an operator such as *before*, *after*, *equal*, and *overlap*. These operators return Boolean values and are used in expressing temporal queries in temporal calculus. Relative time expression could be achieved using Allen's relations such as

before, *after*, *meets*, *overlaps*, and *contains* (Allen, 1983). Recently interval sequence data came into the focus of knowledge discovery process. In contrast to event sequences, interval sequences contain labeled events with a temporal extension. Each event has a label and a timestamp. These temporally extended events are called temporal intervals. Each temporal interval can be described by a triplet (*b*, *e*, *l*), where *b* and *e* denote the beginning and the end of the interval respectively, and *l* represents its label.

Temporal intervals are used to define interval sequences. Without temporal extension there are only two possible relations. Either one event is *before* (or *after*) the other, or the events *coincide*. Due to the temporal extension of temporal intervals, the possible relations between two intervals become more complex. There are seven possible relations between two events. Based on the problems, the interval sequences may have gaps. A temporal interval relation is defined as $R(x,y) = \{P(x,y) \mid (x,y) \in \Omega, P \in IO\}$ (Lee et al., 2009). The set of temporal interval operators *IO* = {*before*, *equals*, *meets*, *overlaps*, *during*, *starts*, *finishes*}, $\Omega = \{(x,y) \mid x,y \in IE, x \neq y\}$ where *IE* is a set of temporal intervals and *P*(*x*, *y*) is a binary predicate which expresses the temporal interval relationship *P* between *x* and *y*.

1.7 Temporal data mining tasks

The objectives of data mining, called *tasks* of data mining, can be classified broadly into some groups (Han & Kamber, 2006; Hand et al., 1999). In case of temporal data mining, the tasks may be grouped as prediction, clustering, classification, searching and retrieval, and pattern discovery.

1.7.1 Prediction

Prediction is often the ultimate goal of temporal data mining. It has variety of applications in the diverse areas such as financial forecasting, meteorology, seismology, and medical disease detection. For example, a couple of specific applications are given as follows: (i) a company is interested in predicting its sales for the next month, and (ii) a doctor would like to predict the reaction of his patients to a new diabetes medication. The first example falls under the area of time series prediction (forecasting), while the second one falls under the category of event prediction. In time series forecasting data are historical, and obtained at regular time intervals. In univariate time series forecasting, the problem is to predict the value of a variable at multiple time intervals. For event prediction, the population data are about the variables that are related to the occurrence of an event or series of events. It predicts the occurrence of an event or the number of occurrences of an event or the duration of an event at given conditions. Predictive model permits the value of one variable to be predicted from the known values of other variables. The goal of temporal prediction is to predict some fields based on other fields. Temporal data prediction also involves using prior temporal patterns such as models and knowledge, for finding the relevant data attributes of interest.

1.7.2 Clustering

Clustering is a process of dividing objects into different groups. It is the subject of active research in several fields such as statistics, pattern recognition, and machine learning. Temporal clustering targets separating temporal data into subsets of similar data. There are two fundamental problems of temporal clustering viz., (i) to define a meaningful similarity measure, and (ii) to choose the number of temporal clusters. In temporal data analysis, many temporal data

mining applications make use of clustering according to similarity and optimization of temporal set functions. If the number of clusters is given then clustering techniques can be divided into three classes: metric-distance based technique, model-based technique, and partition-based technique. These techniques can be used occasionally in combination. For example, in probability-based versus distance-based clustering analysis a combination technique is used. If the number of clusters is not given, then one can use non-hierarchical clustering algorithms to find the number of clusters.

To consider spatial information in clustering, three types of clustering analysis have been studied including spatial clustering (i.e., clustering of spatial points), regionalization (i.e., clustering with geographic contiguity constraints), and point pattern analysis (i.e., hot spot detection with spatial scan statistics). In case of spatial clustering, the similarity between data points or clusters is defined with spatial properties (such as locations and distances). Spatial clustering methods can be partitioning or hierarchical, density-based or grid-based. Today, a huge amount of data are being collected with spatial and temporal components from sources such as meteorological, and satellite imagery. Efficient visualization as well as discovery of useful knowledge from these datasets is therefore very challenging and becoming a massive economic need.

1.7.3 Classification

Data classification is an important machine learning problem with a wide variety of applications. A number of classification models and techniques have been proposed, and applied to the analysis of various datasets. These include methods such as decision trees, bayesian networks, nearest-neighbor classifiers, or support vector machines. However, the success of classification

methods depends heavily on the quality of data and data features used by the models. Consequently, feature selection and feature construction methods in combination with a classification model often determine the success of the machine learning approach in extracting useful and accurate classification models. Advances in data collection and data storage technologies have led to the emergence of complex multivariate datasets. The examples based on these datasets are not simple data points. They even trace the complex behaviors characterized by time series. Consider a problem of building a classifier to diagnose or predict a patient's condition using past patient's data. Ignoring the temporal aspect of data, the patient case can be easily described using the most recent set of values such as "a low blood pressure", or "a high white blood cells count". However, this information may be limited in describing the patient's state. For example, the information that is important for the diagnosis may include simple trends, such as "increase in the blood pressure" or more complex temporal patterns such as "low blood pressure following the prescription of a certain medication". Clearly, more complex temporal information may improve the ability to diagnose the case effectively.

1.7.4 Search and retrieval

Searching for sequences in large databases is another important task in temporal data mining. Sequence search and retrieval techniques play an important role in interactive explorations of knowledge in large sequential databases. The problem is concerned with efficiently locating subsequences, often referred to as queries, in large archives of sequences or, in a single long sequence. Query-based searches have been extensively studied in language and automata theory. While the problem of efficiently locating exact matches of substrings is a well solved problem, but the situation is quite different in looking for *approximate* matches (Wu & Manber, 1992).

For similarity searching in multimedia data, we consider here two main families of multimedia indexing and retrieval systems: (i) description-based retrieval systems, which build indices and perform object retrieval based on image description, such as keywords, captions, size, and time of creation, (ii) content-based retrieval systems, which support retrieval based on the image content, such as colour histogram, texture, shape, objects and wavelet transforms. In content-based retrieval, it is approximate matching that one might be more interested in.

1.7.5 Pattern discovery

Unlike in search and retrieval applications, in pattern discovery there is no specific query to search a database. Pattern discovery is an unsupervised or a supervised operation that is meant only for data mining methods. The main goal of pattern discovery is to discover all interesting patterns in data as discussed in Section 1.6. There is no universal notion for defining a pattern. One of the primary aims of pattern detection is spotting fraudulent behavior by detecting regions of the space defining different types of transactions, where the data points significantly different from the rest.

1.8 Conclusion

A vast amount of temporal data is available in science, engineering and medical fields. Also, many large companies collect data for a long period of time. Knowledge extracted from such data would help the companies to make effective decisions. Due to the existence of a large class of temporal datasets, applications dealing with temporal patterns seem to be present everywhere (Bettini et al., 2000; Lattner, 2007; Mitsa, 2010). In the recent time, some amount of work has been reported on mining and analyzing time-stamped database (Mahanta et al., 2008; Hong et al.,

2009; Khan et al., 2010; Chen & Chundi, 2011). But the domain of multiple time-stamped databases has not been studied well. We have also worked based on this domain. We conclude this chapter by mentioning our contributions made in the thesis.

We have mined various patterns from a collection of yearly databases. These patterns are useful in making many strategic decisions for a company. In Chapter 2, we have proposed a model of mining global temporal patterns in multiple databases, and the notion of degree of stability of an item to extract stable items. We have designed an algorithm for clustering items in multiple databases based on degree of stability. In Chapter 3 we have proposed the concept of generalized notch, and then a special generalized notch, called iceberg, has been proposed. We designed an algorithm to mine icebergs in time-stamped databases. In Chapter 4 we have extended the concept of certainty factor by incorporating support information for an effective analysis of overlapped intervals. We have proposed improvements on the existing algorithm for identifying calendar-based periodic pattern in a time-stamped dataset. We have also analysed the constraints viz., minimum interval, minimum support and maximum gap associated with each time interval. We also provide a comparative analysis between the proposed algorithm and the existing algorithm for mining calendar-based periodic patterns. In Chapter 5 we introduce the notion of an overall influence of a set of items on another set of items. We have proposed an extension to the notion of overall association between two items in a database. Using the notion of overall influence, we have designed two algorithms for influence analysis involving specific items in a database.

Chapter 2

Clustering Items in Different Data Sources Induced by Stability

2.1 Introduction

Due to the liberalization of government policies across the globe, the number of multi-branch companies is increasing over time. Many multi-branch companies deal with multiple databases. Thus, the study of data mining on multiple databases is an important issue. Data mining and knowledge discovery on multiple databases has been recently recognized (Wu & Zhang, 2003; Adhikari & Rao, 2008a) as an important area of research in data mining community.

Many of these multi-branch companies also deal with transactional time-stamped data. Transactional data collected over time at no particular frequency is called *transactional time-stamped data* (Leonard & Wolfe, 2005). Some examples of transactional time-stamped data are point of sales data, inventory data, and trading data. Little work has been reported on the area of mining multiple transactional time-stamped databases. Many important and useful applications might involve transactional time-stamped data.

All the transactions in a branch might get stored locally. A transaction could be viewed as a collection of items with a unique identifier. An interesting characteristic of an item is its variation of sales over time. The items having less variation of sales over time are useful in devising strategies for a company. Thus, it is important to study such items.

Example 1. Let us consider ten yearly time databases and sample time series of supports corresponding to five different items. Let *i*-th series be the time series of supports corresponding to item x_i, for $i = 1, 2, 3, 4, 5$.

(1) 0.03, 0.20, 0.31, 0.11, 0.07, 0.35, 0.82, 0.62, 0.44, 0.13

(2) 0.19, 0.20, 0.18, 0.21, 0.20, 0.20, 0.19, 0.18, 0.21, 0.20

(3) 0.05, 011, 0.07, 0.20, 0.16, 0.12, 0.13, 0.08, 0.17, 0.10

(4) 0.03, 0.04, 0.03, 0.07, 0.08, 0.12, 0.09, 0.15, 0.17, 0.12

(5) 0.04, 0.04, 0.03, 0.05, 0.04, 0.06, 0.04, 0.05, 0.06, 0.05

Among the support series corresponding to different items, we observe that the variation of sales corresponding to item x_5 is the least. The company would prefer to devise strategy based on item x_5. In this chapter we have proposed a model of mining global temporal patterns in multiple databases, and the notion of degree of stability of an item. We have designed an algorithm for clustering items in multiple databases based on the degree of stability.

The rest of the chapter is organized as follows. We discuss related work in Section 2.2. In Section 2.3, we propose a model of mining multiple transactional time-stamped databases. We state our problem in Section 2.4. In Section 2.5, we design an algorithm for clustering of items in multiple databases. Experimental results are provided in Section 2.6.

2.2 Related work

Liu et al. (Liu et al., 2001) have proposed stable association rules based on testing of hypothesis. In this case, the distribution of test statistic under null hypothesis is normal for large sample size. Thus, the stable association rules are determined based on some assumptions. Due to these reasons, we define stable items based on the concept of stationary time series data (Brockwell & Davis, 2002). In the context of interestingness measures, Tan et al. (2002) have described several key properties of twenty one interestingness measures proposed in statistics, machine learning and data mining literature. Wu et al. (2005) have proposed two similarity measures for clustering a set of databases. Zhang et al. (1997) have proposed an efficient and scalable data clustering method BIRCH based on a new in-memory data structure called CF-tree.

Estivill-Castro and Yang (2004) have proposed an algorithm that remains efficient, generally applicable, multi-dimensional but is more robust than to noise and outliers. Jain et al. (1999) have presented an overview of pattern clustering methods from a statistical pattern recognition perspective, with a goal of providing useful advice and references to fundamental concepts accessible to the broad community of clustering practitioners. Authors discussed various clustering processes where objects were considered from single database. In this chapter, we cluster items in multiple databases based on supports of items. Thus, the above algorithms might not be suitable under this framework. Yang and Shahabi (2005) have proposed an algorithm to determine the stationarity of multivariate time series data for improving the efficiency of many correlation based data analysis. Zhang et al. (2003) designed a local pattern analysis for mining multiple databases. Zhang et al. (2004) studied various issues such as data preparation, data privacy related to multi-database mining. A good insight into mining aspect of both single and multi-databases was provided.

2.3 A model of multiple transactional time-stamped databases

Consider a multi-branch company that has n branches. Let D_i be the transactional time-stamped database corresponding to i-th branch, for $i = 1, 2, \ldots, n$. Web sites and transactional databases contain a large amount of time-stamped data related to an organization's suppliers and / or customers over time. Mining these types of time-stamped data could help business leaders make better decisions by listening to their suppliers or customers via their transactions collected over time (Leonard & Wolfe, 2005). We propose a model for mining global patterns in multi-databases over time. Adhikari and Rao (2008b) have proposed an extended model of mining multiple databases using local pattern analysis.

The limitation of this model is that it provides approximate global pattern. Thus, we propose a new model of mining global patterns in multiple transactional time-stamped databases. The proposed model in Figure 2.1 has a set of interfaces and a set of layers. Each interface is a set of operations that produces dataset(s) (or, knowledge) based on the lower layer dataset(s). There are five distinct interfaces of the proposed model of synthesizing global patterns from local patterns. The function of each interface is described below. Interface 2/1 cleans / transforms / integrates / reduces data at the lowest layer. By applying these procedures we get processed database from the original database. In addition, interface 2/1 applies a filtering algorithm on each database for separating relevant data from outlier data. E.g., if we are interested in studying the durable items then the transactions containing only non-durable items could be treated as outlier transactions. Also, it loads data into the respective data warehouse. At interface 3/2, each processed database PD_i is partitioned into k time databases DT_{ij}, where DT_{ij} is the processed database (if available) for the j-th time slot at the i-th branch, for $j = 1, 2, \ldots, k$, and $i = 1, 2, \ldots, n$. At interface 4/3 the j-th time databases of all branches are merged into a single time database DT_j, for $j = 1, 2, \ldots, k$. A traditional data mining technique could be applied on database DT_j at the interface 5/4, for $j = 1, 2, \ldots, k$. Let PB_j be pattern base corresponding to the time database DT_j, for $j = 1, 2, \ldots, k$. Finally, all the pattern bases are processed for synthesizing knowledge or, making decision at the interface 6/5. Other undirected lines in Figure 2.1 are assumed to be directed from bottom to top. The proposed model of mining global patterns over time is efficient, since we get the exact global patterns in multiple databases over time. In layer 4, we have collection of time databases. If any one of these databases is too large to apply a traditional data mining technique then this data mining model would fail. In this situation, we could apply an appropriate sampling

technique to reduce the size of a database. Thus, we get approximate patterns over time.

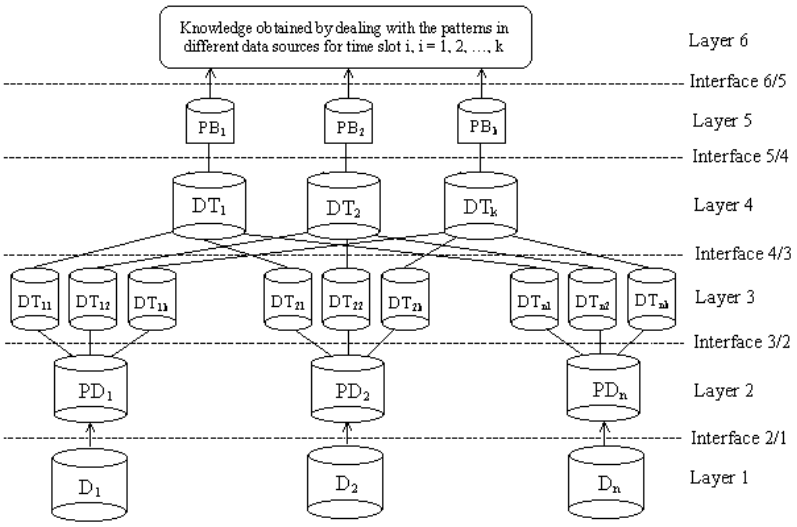

Figure 2.1 A model of mining global patterns in multiple transactional time-stamped databases

2.4 Problem statement

With reference to Figure 2.1, let DT_j be the database corresponding to the *j*-th year, for $j = 1, 2, \ldots, k$. Each of these databases corresponds to a specific period of time. Thus, we could call them as time databases. Each of these time databases is mined using a traditional data mining technique (Han et al., 2000; Agrawal & Srikant 1994). For the specific requirement of this problem, we need to mine only items in the time databases. Let *I* be the set of all items in these databases. Each itemset *X* in a database *D* is associated with a statistical measure, called *support* (Agrawal et al., 1993) denoted by *supp*(*X*, *D*). The support of an itemset is defined as the fraction of transactions containing the itemset. The variation of sales of an item over the time is an

important issue in determining stability of the item. Stable items are useful in many applications, eg. stable items could be useful to promote sales of other items. Modeling with stable item is more justified than modeling with unstable item.

Let $\mu_{s(x)}(t)$ be the mean support of item x in the database DT_1, DT_2, …, DT_t. Thus, $\mu_{s(x)}(t)$ is obtained by the following formula:

$$\mu_{s(x)}(t)=\left(\sum_{i=1}^{t} supp(x, DT_i)\times size(DT_i)\right)\Big/\sum_{i=1}^{t} size(DT_i), \text{ for } t=1, 2, \ldots, k \quad \ldots (1)$$

Let $\sigma(\mu_{s(x)})$ be the standard deviation of $\mu_{s(x)}(t)$, for t = 1, 2, …, k. We call $\sigma(\mu_{s(x)})$ as the *variation of means* corresponding to support of x. Let $\gamma_{s(x)}(t, t + h)$ be the autocovariance of $supp(x, DT_t)$ at lag h, for t = 1, 2, …, k-1. Thus, $\gamma_{s(x)}(t, t + h)$ is obtained by the following formula:

$$\gamma_{s(x)}(t,\ t+h) = \frac{1}{k}\sum_{t=1}^{k-h}\left(supp\left(x, DT_t\right)-\mu_{s(x)}(k)\right)\left(supp\left(x, DT_{t+h}\right)-\mu_{s(x)}(k)\right) \quad \ldots (2)$$

$\sigma(\gamma_{s(x)}(t, t + h))$ be the standard deviation of $\gamma_{s(x)}(t, t + h)$, for h = 1, 2, … , k-1. We call this $\sigma(\gamma_{s(x)}(t, t + h))$ as *variation of autocovariances* corresponding to support of x. We have chosen standard deviation as a measure of dispersion (Bluman, 2006). Standard deviation and mean deviation about mean are relevant measures of dispersion. These measures take into account of variation due to each support unlike the measure range. Skewness, being a descriptive measure of dispersion is not suitable in this context. Before we define stability of an item, we study the following time series of supports corresponding to an item. In the following example, we compute $\sigma(\mu)$ and $\sigma(\gamma)$ of support series corresponding to different items.

Example 2. We continue with Example 1. The variations of means and autocovariances of above series are given as follows: (1) $\sigma(\mu)$ = 0.09342, $\sigma(\gamma)$ = 0.01234, (2) $\sigma(\mu)$ = 0.00230,

$\sigma(\gamma)$ = 0.00002, (3) $\sigma(\mu)$ = 0.02351, $\sigma(\gamma)$ = 0.00039, (4) $\sigma(\mu)$ = 0.02114, $\sigma(\gamma)$ = 0.00076, (5) $\sigma(\mu)$ = 0.0027986, $\sigma(\gamma)$ = 0.0000124. We observe that the value of total variation, $\sigma(\mu) + \sigma(\gamma)$, is the least corresponding to item x_5.

We define stable items based on the concept of stationary time series data (Brockwell & Davis, 2002). In finding $\sigma(\mu)$, we first compute a set of means of support values. Then we compute standard deviation of these mean values. Thus, we find standard deviation of a set of fractions. In finding $\sigma(\gamma)$, we first compute a set of autocovariances of support values. Then we compute standard deviation of these autocovariances. An autocovariance of supports is an average of a set of squared fractions. Thus, we find standard deviation of a set of squared fractions. So, $\sigma(\mu) \geq \sigma(\gamma)$. In fact, $\sigma(\gamma)$ is close to 0. Thus, we define our first measure of stability $stable_1$ as follows.

Definition 1. An item x is stable if $\sigma(\mu_{s(x)}) \leq \delta$, where δ is user defined *maximum threshold*.

More strictly, we may wish to impose restrictions on both $\sigma(\mu)$ and $\sigma(\gamma)$. Thus we define our second measure of stability $stable_2$ as follows.

Definition 2. An item x is stable if $\sigma(\mu_{s(x)}) + \sigma(\gamma_{s(x)}) \leq \delta$, where δ is user defined maximum threshold.

In Definition 2, the expression $\sigma(\mu) + \sigma(\gamma)$ is the determining factor of stability of an item. We define *degree of variation* of an item x as follows.

$$degOfVar(x) = \sigma(\mu_{s(x)}) + \sigma(\gamma_{s(x)}) \quad \ldots (3)$$

Higher value of *degOfVar* implies lower degree of stability of the item. Based on above discussion, we state our problem as follows.

Let D_i and DT_j be the databases corresponding to i-th branch and j-th year of a multi-branch company as depicted in Figure 1 respectively, for i = 1, 2, …, n, and j = 1, 2, …, k. Each of the time (year) databases has been mined using a traditional data mining technique. Based on the mining results, degree of variation of each item has been computed as discussed above. Find the best non-trivial partition (if it exists) of the items in D_1, D_2, …, D_n based on degree of variation of an item.

A partition (Liu, 1985) is a specific type of clustering. Formal definition of non-trivial partition is given in Section 2.5.

2.5 Clustering items

Our clustering technique is based on the notion of degree of stability of an item. Again, the degree of stability is based on the variations of means and autocovariances. The clustering technique requires computing the degree of variation for each item in the databases. Let *I* be the set of all items in the databases. Given a set of yearly databases, the difference in variations between every pair of items could be expressed by a square matrix, called *difference in variation* (*diffInVar*). We construct *diffInVar* as follows.

$$diffInVar(i, j) = |\, degOfVar\,(x_i) - degOfVar(x_j)\,|, \text{ for } x_i, x_j \in I. \quad \ldots (4)$$

In the following example, we compute *diffInVar* corresponding to Example 1.

Example 3. We continue here with Example 1. Matrix *diffInVar* is given as follows.

$$diffInVar = \begin{bmatrix} 0 & 0.103 & 0.082 & 0.084 & 0.102 \\ 0.103 & 0 & 0.021 & 0.019 & 0.001 \\ 0.082 & 0.021 & 0 & 0.002 & 0.020 \\ 0.084 & 0.019 & 0.002 & 0 & 0.018 \\ 0.102 & 0.001 & 0.020 & 0.018 & 0 \end{bmatrix}$$

Matrix *diffInVar* is symmetric square matrix. We shall use this matrix for clustering items in multiple databases.

Intuitively, if the difference in variations between two items is close to zero then they may be put in the same class. Before clustering the items, we define a class as follows.

Definition 3. Let $I = \{i_1, i_2, \ldots, i_p\}$ be the set of items. A class formed at the level of difference in variation α is defined as follows.

$$class(I,\alpha) = \begin{cases} X : X \subseteq I, |X| \geq 2, \text{and } degOfVar(x_1, x_2) \leq \alpha, \text{for } x_1, x_2 \in X \\ X : X \subseteq I, |X| = 1 \end{cases}$$

Based on the above definition of a class, we define a clustering as follows.

Definition 4. Let $I = \{i_1, i_2, \ldots, i_p\}$ be the set of items. Let $\pi(I, \alpha)$ be a clustering of items in I at the level of difference in variation α. Then, $\pi(I, \alpha) = \{X: X \in \rho(I)$, and X is a $class(I, \alpha)\}$, where $\rho(I)$ is the power set of I.

During the clustering process we may like to impose the restriction that each item belongs to at least one class. This restriction makes a clustering complete. We define a complete clustering as follows.

Definition 5. Let $I = \{i_1, i_2, \ldots, i_p\}$ be the set of items. Let $\pi(I, \alpha) = \{C_1(I, \alpha), C_2(I, \alpha), \ldots, C_m(I, \alpha)\}$, where $C_k(I, \alpha)$ is the k-th class of the cluster π, for $k = 1, 2, \ldots, m$. π is complete, if $\bigcup_{k=1}^{m} C_k(I,\alpha) = I$.

In a complete clustering, two classes may have common items. We may be interested in finding out a cluster containing mutually exclusive classes. A mutually exclusive cluster could be defined as follows.

Definition 6. Let $I = \{i_1, i_2, \ldots, i_p\}$ be the set of items. Let $\pi(I, \alpha) = \{C_1(I, \alpha), C_2(I, \alpha), \ldots, C_m(I, \alpha)\}$, where $C_k(I, \alpha)$ is the k-th class of the cluster π, for $k = 1, 2, \ldots, m$. π is mutually exclusive if $C_i(I, \alpha) \cap C_j(I, \alpha) = \phi$, for $i \neq j$, and $1 \leq i, j \leq m$.

We may be interested in finding out such a mutually exclusive and complete cluster. A partition of a set of items I is defined as follows.

Definition 7. Let $\pi(I, \alpha)$ be a mutually exclusive and complete cluster of a set of items I at the level of difference in variation α. $\pi(I, \alpha)$ is called a non-trivial partition if $1 < |\pi| < m$.

A partition is a cluster. But a cluster is not necessarily be a partition. In the next section, we find the best non-trivial partition (if it exists) of a set of items. The items in a class are similar with respect to their variations. We are interested in the classes of a partition where the variations of items are less. The items in these classes are useful in devising strategies for the company. Thus, we define average degree of variation *adv*, of a class as follows.

Definition 8. Let C be a class of partition π. Then,

$$adv(C \mid \pi) = \frac{1}{|C|} \Sigma_{x \in C} degOfVar(x).$$

2.5.1 Finding the best non-trivial partition

With reference to Example 1, we arrange all non-zero and distinct values of *diffInVar* in non-decreasing order for finding all the non-trivial partitions, for $1 \leq i < j \leq 5$. The arranged values of *diffInVar* are given as follows: 0.001, 0.002, 0.018, 0.019, 0.020, 0.021, 0.082, 0.084, 0.102, 0.103. We get two non-trivial partitions at α = 0.001, and 0.002. The partitions are given as

follows: $\pi^{0.001}$ = {{x_1}, {x_2, x_5}, {x_3}, {x_4}}, and $\pi^{0.002}$ = {{x_1}, {x_2, x_5}, {x_3, x_4}}. We observe that at different levels of α we have different partitions. We would like to find the best partition among these partitions. The best partition is based on the principle of minimizing the intra-class variation and minimizing the inter-class similarity. Intra-class variation and inter-class similarity are defined as follows.

Definition 9. The intra-class variation *intra-var* of a partition π at the level α is defined as follows.

$$intra\text{-}var(\pi^{\alpha}) = \Sigma_{k=1}^{|\pi|} \Sigma_{x_i, x_j \in C_k; x_i < x_j} | degOfVar(x_i) - degOfVar(x_j) |$$

Definition 10. The inter-class similarity *inter-sim* of a partition π at the level α is defined as follows.

$$inter\text{-}sim(\pi^{\alpha}) = \Sigma_{c_p, c_q \in \pi; p<q} \Sigma_{x_i \in C_p, x_j \in C_q} \text{minimum}\{degOfVar(x_i), degOfVar(x_j)\}$$

The best partition among a set of partitions is selected on the basis of goodness value of a partition. Goodness measure *goodness*, of a partition is defined as follows.

Definition 11. The goodness of a partition π at level α is defined as follows: $goodness(\pi^{\alpha})$ = $intra\text{-}var(\pi^{\alpha}) + inter\text{-}sim(\pi^{\alpha}) - |\pi^{\alpha}|$, where $|\pi^{\alpha}|$ is the number of classes of π.

We have subtracted $|\pi^{\alpha}|$ from the sum of intra-class variation and inter-class similarity to remove the bias of goodness value of a partition. Better partition is obtained at higher goodness value. We would like to partition the set of items in Example 1 using above goodness measure.

Example 4. With reference to Example 2, we calculate goodness value of each of the non-trivial partitions.

intra-var($\pi^{0.001}$) = 0.001, *inter-sim*($\pi^{0.001}$) = 0.081, and $|\pi^{0.001}|$ = 4. Thus, *goodness*($\pi^{0.001}$) = -3.916.

intra-var($\pi^{0.002}$) = 0.003, *inter-sim*($\pi^{0.002}$) = 0.06, and $|\pi^{0.002}|$ = 3. Thus, *goodness*($\pi^{0.002}$) = -2.937.

The goodness value corresponding to the partition $\pi^{0.002}$ is the maximum. Thus, the partition $\pi^{0.002}$ is the best among the non-trivial partitions. Let us return back to Example 1. There are five series of supports corresponding to five items. Based on variation among the supports in a series, we could partition the series as follows: {series 1}, {series 2, series 5}, {series 3, series 4}. Hence, we get the following partition: $\{x_1\}$, $\{x_2, x_5\}$, $\{x_3, x_4\}$. The proposed clustering technique also identifies the same partition as the best partition. Thus, it verifies the correctness of the proposed clustering technique. *adv* ($\{x_1\}| \pi^{0.002}$) = 0.105, *adv* ($\{x_2, x_5\}| \pi^{0.002}$) = 0.0025 and *adv* ($\{x_3, x_4\}| \pi^{0.002}$) = 0.022. We find that the average degree of variation of $\{x_2, x_5\}$ is the least among the classes of $\pi^{0.002}$. Thus, the items x_2 and x_5 are most suitable among all the items in the given databases for making strategies of the company.

We design an algorithm for finding best non-trivial partition of items in multiple databases. First we describe different data structures used in designing an algorithm for finding the best partition of items. For each item there are k supports corresponding to k different years. We maintain $m \times k$ supports for m items in array *supports*. The i-th row of *supports* stores supports corresponding to i-th item for k years, for $i = 1, 2,\ldots, m$. Let *means* be a two dimensional array such that the i-th row stores means of supports corresponding to different years for i-th item, for $i = 1, 2,\ldots, m$. Let *autocovariances* be a two dimensional array such that the i-th row stores autocovariances of supports corresponding to different lags for i-th item, for $i = 1, 2,\ldots, m$. For

year *j*, we compute mean value of supports for year 1 to *j*. Thus we get different mean values for different years. Let *stdDevMeans* be the standard deviation of these mean values. For year *j*, we also compute autocovariances of supports for year 1 to *j* at different lags. Thus we get different autocovariances for different lags corresponding to a year. Let *stdDevAutocovars* be the standard deviation of these autocovariances. The degrees of variation of different items are stored in array *degInVar*. Variable *S* is a one dimensional array containing ${}^{m}C_2$ difference in variations. *adv* is a one dimensional array which stores the average degree of variation for the items in each class. The algorithm is presented below.

Algorithm 1. Find best non-trivial partition (if it exists) of items in multiple databases.

procedure B*estPartition* (*m*, *supports*)

Inputs:

m: number of items

supports: array of supports of different items corresponding to different years

Outputs:

Best non-trivial partition (if it exists) of items in multiple databases

01: **for** *i* = 1 to *m* **do**

02: compute *means*(*i*) using formula (1) at different years;

03: **let** *stdDevMeans* = standard deviation of mean values for different years;

04: compute *autocovariance*(*i*) using formula (2) at different time lags;

05: **let** *stdDevAutocovars* = standard deviation of autocovariances;

06: compute *degOfVar*(*i*) = *stdDevMeans* + *stdDevAutocovar*;

07: **end for**

08: **for** *row* = 1 to *m* **do**

09: **for** *col* = (*row* + 1) to *m* **do**

10: compute *diffInVar*(*row*, *col*) using formula (4);

11: **end for**

12: **end for**

13: sort distinct elements in the upper triangle of *diffInVar* in non-decreasing order into *S*;

14: **let** *k* = 1; **let** *maxGoodness* = -9999; $\pi = \phi$;

15: **while** (*k* ≤ |*S*|) **do**

16: **let** *curRow* = 1; **let** *curClass* = 1;

17: **for** *i* = 2 to *m* **do**

18: *classLabel*(*i*) = 0;

19: **end for**

20: **let** *classLabel*(1) = 1;

21: **let** *curDiffVar* = *S*(*k*);

22: **for** *col* = *curRow* + 1 to *m* **do**

23: **if** (*diffInVar*(*curRow*, *col*) ≤ *curDiffVar*) **then**

24: **if** (*classLabel* (*col*) = 0) **then**

25: *classLab*el(*col*) = *curClass*;

26: **else if** (*classLabel* (*col*) ≠ *curClass*) **then**

27: partition does not exist at this level;

28: go to line 49;

29: **end if**

30: **end if**

31: **end for**

32: increased *curRow* by 1;

33: **if** (*classLabel*(*curRow*) = 0) **then**

34: increase *curClass* by 1;

35: *classLabel*(*curRow*) = *curClass*;

36: **else** *curclass* = *classLabel*(*curRow*);

37: **end if**

38: **if** (*curRow* ≤ *m*) go to line 22; **end if**

39: **let** *j* = 0;

40: **while** ((*classLabel*(*j*) ≠ 0) **and** (*j* < *m*)) **do**

41: increase *j* by 1;

42: **end while**

43: **if** (*j* = *m* + 1) **then**

44: **if** (*maxGoodness* < goodness value of current partition) **then**

45: *maxGoodness* = goodness value of current partition;

46: store current partition into π;

47: **end if**

48: **end if**

49: increase *k* by1;

50: **end while**

51: return π;

end procedure

In this paragraph, we explain different lines of above algorithm. Algorithm 1 computes *degreeOfVar* for all items using lines 1-7. Matrix *diffInVar* is constructed using lines 8-12. We check the existence of partition at every value in *S*. We start checking partition by assigning the first item to *classLabel* 1. Also, clustering process is performed row by row, starting from row number 1. At the *i*-th row, all the items greater than *i* are classified. During this process, if a labeled item gets another label then we conclude that partition does not exist at the current level. After increasing the current row by 1 we check the class label corresponding to current row. Each row corresponds to an item in the database. If the current row is not labeled yet then we increase the class label by1. If the goodness value of the current partition is less than the *goodness* value of another partition then the current partition is ignored.

Lemma 1. Algorithm 1 executes in $O(m^4)$ time.

Proof. Line 2 takes $O(k)$ time to compute $means(i)$, for some $i = 1, 2, \ldots, m$. Also, line 3 takes $O(k)$ time to compute standard deviation of mean values. To compute formula (2), we require $O(k)$ time. Thus, line 4 takes $O(k^2)$ time. In line 5, we compute standard deviation of k-1 autocovariance values. Thus, line 5 takes $O(k^2)$ time. The for-loop in lines 1-7 repeats m times. Thus, the for-loop in lines 1-7 take $O(m \times k^2)$ time. For computing *diffInVar* at a given row and column, it takes $O(1)$ time. Thus, lines 8-12 take $O(m^2)$ time. There are maximum ${}^{m-1}C_2$ elements in the upper triangle of *diffInVar*. Thus, line 13 takes $O(m^2 \times \log(m))$ time. The while-loop at line 15 repeats maximum ${}^{m-1}C_2$ times. Each of the loops at lines 17, 22, and 40 takes $O(m)$ time. To store a partition it takes $O(m)$ time. To compute goodness value for a particular partition, it takes

$O(m^2)$ time. Thus, the lines 15-50 take $O(m^4)$ time. The time complexity of *bestPartition* algorithm is $O(m^4)$.

In finding stable items in multiple databases, a class having minimal average degree of variation in the best partition might not be a best class at a given degree of stability. In many applications, we may need to find stable items at a given degree of stability. In this case, it might not be a requirement that the stable items need to form a class of a non- trivial partition. Thus, the question of finding a partition might not arise always. To find such a class we shall follow a different approach.

2.5.2 Finding best class

Before finding best class, we first define the concept of best class as follows.

Definition 12. Let C be a class of items. C is called a best class at the level of difference in variation α if (i) $|degOfVar(x) - degOfVar(y)| \leq \alpha$, for $x, y \in C$, (ii) $adv(C)$ is the minimum among all classes of maximal size, and (iii) C has a maximal size.

In Lemma 1, we show that it might not be possible to find two classes of maximal size having the same average degree of variation.

Lemma 2. Best class is unique.

Proof. Let $x_1, x_2,\ldots, x_m$ be the items sorted on non-decreasing degree of variation. We conclude that item x_1 has maximum stability, and the item x_m has minimum stability. At level α, let the stabilities of items $x_1, x_2,\ldots, x_k$ be less than or equal to α, and the stabilities of items $x_{k+1}, x_{k+2}, \ldots, x_m$ be greater than α, for $1 \leq k \leq m$. The best class has least average degree of variation. Also, the difference in variation of two items in the class is less than or equal to α. Thus, $\{x_1, x_2,\ldots, x_k\}$

forms the best class. We are not concerned whether it becomes a member of a partition. *adv*($\{x_1, x_2, \ldots, x_k\}$) is the minimum, and hence best class is unique.

We might be interested in finding best class of items in multiple databases. We use array *class* to hold the best class of items. In the following, we provide an algorithm in finding best class of items in multiple databases.

Algorithm 2. Find the best class of items in multiple databases induced by stability.

procedure B*estClass* (*m, α, supports*)

Inputs:

m: number of items

α: level of degree of variation

supports: array of supports of different items corresponding to different years

Outputs:

Best class of items in multiple databases

01: perform lines 01 – 07 of Algorithm 1;

02: sort array *degOfVar* in non-decreasing order;

03: **let** *class* (1) = *degOfVar*(1); **let** *count* = 1; **let** *avgVar* = 0;

04: **for** *i* = 2 to *m* do

05: *class* (*i*) = -1;

06: **end for**

07: **for** *i* = 2 to *m* **do**

08: **if** ((*degOfVar*(*i*) – *degOfVar*(1)) ≤ α)

09: *class* (*i*) = *degOfVar*(*i*);

10: increase *count* by 1;

11: *avgVar* = *avgVar* + *degOfVar*(*i*);

12: **end if**

13: **end for**

14: *avgVar* = *avgVar* / *count*;

15: return (*class*, *count*, *avgVar*);

end procedure

In this paragraph, we explain different lines of above algorithm. We compute degree of variations for all items using lines 1-7 and store them in array *degreeOfVar* in non-decreasing order. The best class would contain the first item of *degreeOfVar*. The item with least *degreeOfVar* is assigned to *class* 1. An item *i* is included in the best class if (*degOfVar*(*i*) – *degOfVar*(1)) ≤ α. Algorithm 2 returns best class *class*, the number of items in the best class *count*, and the average degree of variation of the best class *avgVar*.

Lemma 3. Algorithm 2 executes in maximum { $O(m \times k^2)$, $O(m \times \log(m))$ }time.

Proof. Line 1 executes in $O(m \times k^2)$ time [Lemma 2], where k is the number of years. There are two for loops in Algorithm 2 apart from loops placed in line 1. Each of these loops executes in $O(m)$ time. Line 2 takes $O(m \times \log(m))$ time. Thus, the lemma follows.

2.6 Experiments

We have carried out several experiments to study the effectiveness of our approach. All the experiments have been implemented on a 1.6 GHz Pentium IV with 256 MB of memory, using the software visual C++ (version 6.0). We present the experimental results using two real

datasets *mushroom* (Frequent itemset mining dataset repository), and *ecoli* (UCI ML repository). Dataset *ecoli* is a subset of *ecoli database* and it has been processed for the purpose of conducting experiments. Let *DB*, *NT*, *ALT*, *AFI*, and *NI* denote database, the number of transactions, average length of a transaction, average frequency of an item, and number of items respectively. We present some characteristics of these datasets in Table 2.1. Each dataset has been divided into 10 databases, called input databases, for the purpose of conducting experiments. The input databases obtained from *mushroom* and *ecoli* are named as M_i, and E_i, for $i = 0, 1, \ldots, 9$. The *mushroom* dataset contains 8,124 transactions and 120 distinct items. Its average transactions size is 24, around 20% ((24/120)×100) of its distinct items are present in every transaction and therefore it is a dense dataset. Similarly, density of *ecoli* and *random-68* is 0.077 and 0.08 and therefore, they are considered as sparse. We present some characteristics of the input databases in Table 2.2.

Table 2.1 Dataset characteristics

Dataset	***NT***	***ALT***	***AFI***	*NI*
mushroom	8124	24.000	1624.800	120
ecoli	336	7.000	25.835	91
random-68	3000	5.460	280.985	68

In Table 2.3, we present top 10 stable items in multiple databases. In *mushroom*, 85 has least degree of variation and it is zero. From experiment we observed that item 85 occurs in every transaction and its support is 1.0 in all the input databases. One could also notice that items of *mushroom* has less degree of variation as compare to *ecoli* and *random-68*.

Table 2.2 Time database characteristics

DB	***NT***	***ALT***	***AFI***	***NI***	***DB***	***NT***	***ALT***	***AFI***	***NI***
M_0	812	24.000	295.273	66	M_5	812	24.000	221.454	88
M_1	812	24.000	286.588	68	M_6	812	24.000	216.533	90
M_2	812	24.000	249.846	78	M_7	812	24.000	191.059	102
M_3	812	24.000	282.435	69	M_8	812	24.000	229.270	85
M_4	812	24.000	259.840	75	M_9	816	24.000	227.721	86
E_0	33	7.000	4.62000	50	E_5	33	7.000	3.915	59
E_1	33	7.000	5.133	45	E_6	33	7.000	3.500	66
E_2	33	7.000	5.500	42	E_7	33	7.000	3.915	59
E_3	33	7.000	4.812	48	E_8	33	7.000	3.397	68
E_4	33	7.000	3.397	68	E_9	39	7.000	4.550	60
R_0	300	5.590	28.676	68	R_5	300	5.140	26.676	68
R_1	300	5.417	28.000	68	R_6	300	5.510	28.353	68
R_2	300	5.360	27.647	68	R_7	300	5.497	28.338	68
R_3	300	5.543	28.456	68	R_8	300	5.537	28.471	68
R_4	300	5.533	28.382	68	R_9	300	5.477	28.235	68

Table 2.3 Top 10 stable items in multiple databases

mushroom		*ecoli*		*random-68*	
item	*degOfVar*	item	*degOfVar*	item	*degOfVar*
85	0.000	1	0.0013	42	0.0012
8	0.0002	99	0.0018	41	0.0018
12	0.0003	91	0.0022	37	0.0023
75	0.0003	4	0.0025	67	0.0027
89	0.0003	94	0.0036	11	0.0028
62	0.0009	15	0.0038	45	0.0029
22	0.0010	12	0.0039	18	0.0030
20	0.0011	19	0.0040	56	0.0031
82	0.0012	3	0.0040	3	0.0031
33	0.0014	10	0.0044	28	0.0031

Table 2.3 shows that the range of variation in *mushroom* is (0.0 – 0.0014) for top ten stable items. Whereas the degree of variation range among the items in *ecoli* and *random-68* is (0.0013 – 0.0044) and (0.0012 – 0.00311) respectively. In Table 2.4, we present five best classes and their average degree of variation for a given value of α for each database. Since *mushroom* is dense we get best five classes at α = 0.00025 and at α = 0.0014. We also observe the *Adv* of these classes are quite less. Unlike *mushroom* higher value of α is considered for *ecoli* and *random-68* because they are sparse and *Adv* is also high for all these classes. The items with least degree of variations such as {85, 8} in *mushroom*, {1, 99} in *ecoli* and {42, 41} in *random-68* are included first for lowest α. The best partition contains two classes. The reason behind that the item with highest *degofvar* forms a class and all the remaining items belong to other class.

The best partition of items in *mushroom* dataset is obtained at level 0.2476. The amount of intra variation, inter similarity, and goodness value are 413.59719, 377.58308, and 789.18027 respectively. Here 56 has highest degree of variation with 0.02478 and therefore it is not included in best class. The best class is given as follows: {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, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119 }. It has average degree of variation 0.05682.

The best partition of items in *ecoli* dataset is obtained at level 0.05632. The amount of intra variation, inter similarity, and goodness value are 44.17961, 185.60862, and 227.78823 respectively. Item 35 has highest degree of variation with 0.058126. The best class is given as follows: {0, 1, 3, 4, 5, 6, 7, 8, 10, 11, 12, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 36, 37, 38, 39, 40, 41, 43, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 94, 99, 100}. It has average degree of variation 0.01829.

The best partition of items in *random-68* dataset is obtained at level 0.013670. The amount of intra variation, inter similarity, and goodness value are 4.627779, 18.608446, and 21.236225 respectively. Item 4 has highest degree of variation with 0.0155. The best class is given as follows: {1, 2, 3, 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, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68}.

It has average degree of variation 0.006382. We have studied execution time with respect to number of data sources. We observe in Figures 2.2, 2.3 and 2.4 that the execution time increases as the number of data sources increases. Execution time depends on *NI*, *NT*, *ALT* and number of input databases. In *mushroom* the density of input databases vary from 23% to 33%. Number of transactions of each input databases are 812. Thus, maximum time is taken 2725 sec.

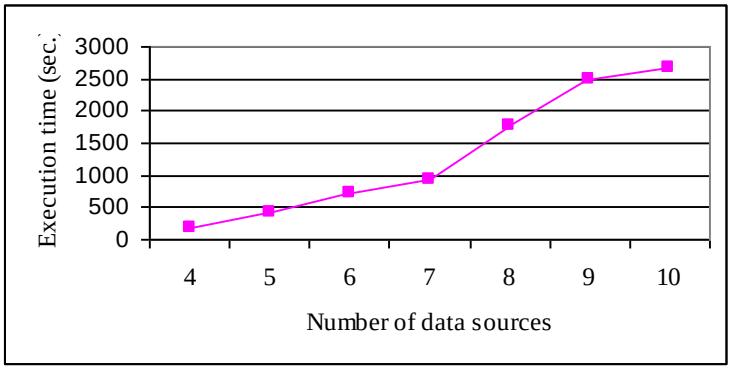

Figure 2.2 Execution time vs. number of data sources obtained from *mushroom*

Table 2.4 5 best classes in multiple databases

mushroom			*ecoli*			*random-68*		
α	items	*adv*	α	items	*adv*	α	items	*adv*
0.00025	{85, 8}	0.0001	.0.0020	{1, 99, 91, 4}	0.0020	0.0010	{42, 41}	0.0015
0.0003	{85, 8, 12, 75, 89}	0.0002	0.0025	{1, 99, 91, 4, 94}	0.0023	0.0015	{42, 41, 37}	0.0018
0.0010	{85, 8, 12, 75, 89, 62}	0.0003	0.0030	{1, 99, 91, 4, 94, 15,12,19, 3}	0.0030	0.0017	{42,41,37,67,11 ,45}	0.0023
0.0012	{85, 8, 12, 75, 89, 62, 22, 20}	0.0005	0.0035	{1, 99, 91, 4, 94, 15,12,19, 3,10,6}	0.0033	0.0020	{42,41,37,67,11 ,45,18,56,3, 28}	0.0026
0.0014	{85, 8, 12, 75, 89, 62, 22,20, 82}	0.0006	0.0050	{1, 99, 91, 4, 94, 15,12,19, 3,10,6,18}	0.0035	0.0022	{42,41,37,67,11 ,45,18,56,3,28,7 ,53}	0.0027

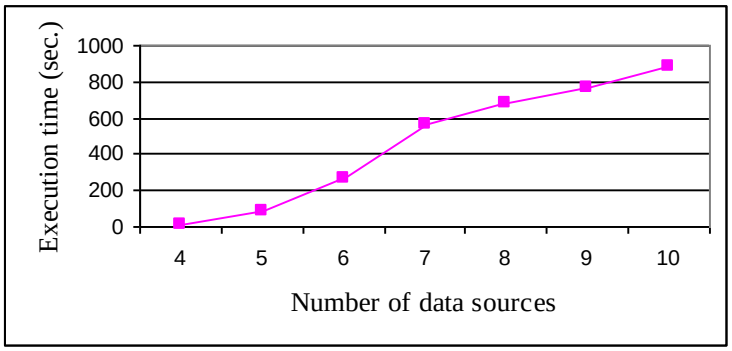

Figure 2.3 Execution time vs. number of data sources obtained from *ecoli*

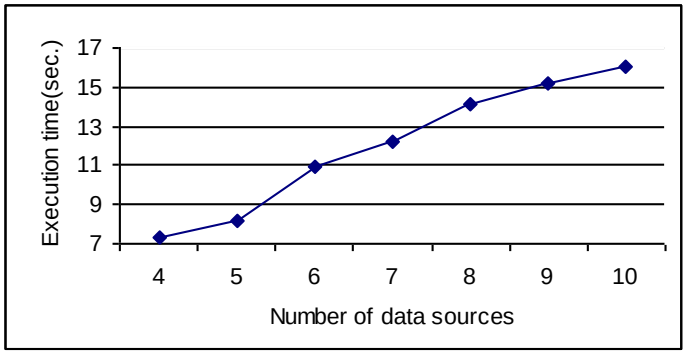

Figure 2.4 Execution time vs. number of data sources obtained from *random-68*

From Figure 2.2 we observe that initially the algorithm takes less time, later there is a sharp rise for every increase number of input databases. The density of input databases in *ecoli* vary from 10% to 17% and 8% in *random-68*. Both the databases are sparse and *ALT* is also low and therefore figures 2.3 and 2.4 show sharp rise of execution time from the very beginning.

2.7 Conclusion

Stable items are useful for modeling various strategies of an organization. Thus, it is necessary to identify stable items. We propose the notion of degree of stability of an item. We design an algorithm for clustering items in multiple databases based on degree of stability. The proposed technique is useful and effective.

Chapter 3

Mining Icebergs in Time-Stamped Databases

3.1 Introduction

Many organizations collect transactional data continuously over a long period of time. A database grown over a long period of time might contain useful as well as interesting temporal patterns. By taking into the account time aspect, many interesting applications / patterns such as surprising patterns (Keogh et al., 2002), discords (Keogh et al., 2006), calendar based patterns (Mahanta et al., 2008) have been reported in the recent time. Surprising patterns, anomaly detection and discords could be considered as exceptional patterns occurring in a time series. These exceptional patterns are important as well as interesting contributions to temporal data mining. In this chapter, we study another kind of exceptional patterns in transactional time-stamped data. We define the exceptional patterns and discuss how to mine them from time-stamped databases.

Though an analysis of time series data (Box et al., 2003; Brockwell & Davis, 2002; Keogh, 1997; Tsay, 1989) has been intensively studied, the analysis of time-stamped data still calls for more research. Specifically, in the context of multiple time-stamped databases, little work has been reported so far. Therefore, there arises an urgent need to study multiple time-stamped databases. In Example 1, we observe an interesting type of temporal pattern in multiple time-stamped databases that needs to be analyzed fully.

The support of an itemset (Agrawal et al., 1993) is defined as the fraction of transactions containing the itemset. It has been used extensively in identifying different types of patterns in a database. Some examples are association rule (Agrawal et al., 1993), negative association rule (Wu et al., 2004) and conditional pattern (Adhikari & Rao, 2008c). Nonetheless the support

measure has a limited use in discovering some other types of patterns in a database. We illustrate this issue using the following example.

Example 1. Consider a company that maintains customers' transactions on a yearly basis. Many important problems can be studied given such yearly databases. Let item *A* be of our interest. In view of analyzing item *A* over the years, let us consider the sales series of *A* from the year 1970 to 1979.

(0.9, 1000, 1970), (0.31, 2700, 1971), (0.36, 2500, 1972), (0.29, 3450, 1973), (0.37, 1689, 1974), (0.075, 7098, 1975), (0.073, 8900, 1976), (0.111, 6429, 1977), (0.09, 9083, 1978), (0.07, 10050, 1979). The first, second, and the third component of each triple refers to the support of *A*, the number of transactions in the yearly database and the corresponding year, respectively.

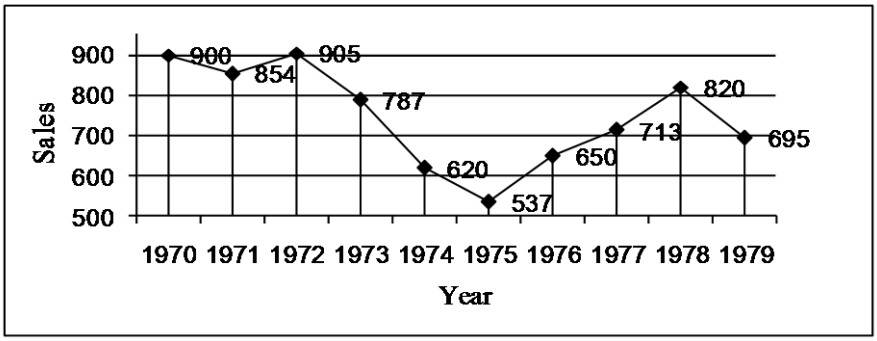

Figure 3.1 Sales of item *A* reported in consecutive years

The sales series of item *A* is depicted in Figure 3.1. There is a significant downfall of sales from 1972 and rise in sales from the year 1975. Year 1975 is an important point (Pratt & Fink, 2002) for the company. It is a significant down-to-up change in the sales series. It is not surprising to observe a significant up-to-down change in a sales series of an item. Such patterns in time-stamped series are interesting as well as important to investigate. They could reveal the sources of customers' purchase behavior and that might provide an important knowledge to the organization.

At this point, one might be interested in knowing how a time series data differs from a time-stamped data. Transactional data are time-stamped data collected over time at no particular frequency (Leonard & Wolfe, 2005). Whereas, time series data are time-stamped data collected over time at a particular frequency. For example, point of sales data could be considered as time-stamped data, but sales per month / year could be considered as time series data. One could convert transactional data into time series data for the purpose of specific data analyses. The frequency associated with time series data varies from problem to problem. For future planning of business activities, one might need to analyze the past data of customer transactions collected over a long period of time. While analyzing the past data it is useful as well as important to figure out the abrupt changes in sales of an item along with time. Existing algorithms, as mentioned above, fail to detect these changes. Therefore, in this chapter our objective is to define such exceptional patterns and design an algorithm to extract such patterns from time-stamped databases.

For the purpose of studying patterns in time-stamped databases one may need to handle multiple databases over time. One could call these time variant databases as *time databases*. In this context, the choice of time granularity is an important issue as the characteristics of temporal patterns is heavily dependent on this parameter. It is quite reasonable to consider the time granularity as one year, since a season re-appears on a yearly basis and the customers' purchase behaviour might vary from season to season.

Consider an established company having data over fifty consecutive years. The company might be interested in knowing the performance of different items over the years. Such analysis might help the company in devising the future strategies.

The objective of this chapter is to identify abrupt changes in sales of each item (as defined in Sections 3.4 and 3.5) over the years as depicted in Figure 3.1. The goal of this chapter is to define a new type of pattern based on abrupt variation of sales of an item over the years and to design an algorithm to mine such patterns in time databases.

Rest of the chapter is organized as follows. We discuss related work in Section 3.2. In Section 3.3, we introduce a new temporal pattern, called notch, of an item. Based on this pattern, we propose the concepts of generalized notch (Section 3.4) and iceberg notch (Section 3.5). We present another view of sales series in Section 3.6. In Section 3.7 we design an algorithm for mining icebergs in time-stamped databases. Experimental results are presented in Section 3.8.

3.2 Related work

Temporal sequences appear in a vast range of domains ranging from engineering to medicine and finance, and the ability to model and extract information from them becomes crucial from a conceptual as well as applied perspective. Identifying exceptional patterns in time-stamped databases deserves much attention. In Sections 3.4 and 3.5 we will propose two exceptional patterns in time-stamped databases.

There are mainly two broad directions of temporal data mining (Roddick & Spillopoulou, 1999). One concerns the discovery of casual relationships among temporally oriented events. Another one deals with the discovery of similar patterns within the same time sequence or among different time sequences. Sequences of events describe the behavior and actions of users or systems that can be collected in several domains.

The proposed problem falls under the first category of problems, since we are interested in identifying exceptional patterns by comparing sales in different years.

Agrawal et al. (1995) introduced the *shape definition language* (SDL), which used limited vocabulary such as {*Up, up, stable, zero, down, Down*} to describe different gradients in the series. The similarity of two time series is proportional to the length of the longest common sequence of words in their *SDL* representation. Such coarse information might not be always helpful. We define two exceptional patterns viz. a generalized notch and an iceberg notch.

Perng et al. (2000) proposed the landmark model where perceptually important points of a time series are used in its representation. The perceptual importance depends on the specific type of the time series. In general, sound choices for landmarks are local maxima and minima as well as inflection points. The advantage of using the landmark-based method is that this time representation is invariant to amplitude scaling and time warping. Some of the local maxima and minima might lead to higher level of exceptionality. Here we are concerned with defining such exceptionalities in time-stamped databases.

There has been a significant amount of work on discovering temporal patterns of interest in sequence databases and time series databases. Temporal data mining is concerned with the analyses of data with an intention of finding patterns and regularities from a set of temporal data. In this context sequential association rule (Agrawal & Shrikant, 1995), periodical association rule (Li & Deogun, 2005), calendar association rule (Li et al., 2003) calendar-based periodic pattern (Mahanta et al., 2008) and up-to-date pattern (Hong et al., 2009) are some of the interesting temporal patterns reported in the recent time.

As noted in Section 3.1, support history of an item provides important information of an item over time. We have proposed an algorithm for clustering items in multiple databases based on their support history (Adhikari et al., 2009). We have introduced the notion of stability of an item based on its support history.

Lomet et al. (2008) integrated a temporal indexing technique, the TSB-tree, into Immortal DB to serve as the core access method. The TSB-tree provides high performance access and update for both current and historical data.

Keogh et al. (2005) proposed an algorithm for finding unusual time series where the notion of time discords is introduced. A time discord is a subsequence of a longer time series that is maximally different from all other subsequences of the series. Discords can be used to detect anomalies in an efficient way.

Many algorithms are designed incrementally to support time-dependent analysis of data. We have proposed algorithms incrementally to study overall influence of a set of items on another set of items in time databases (Adhikari & Rao, 2010).

Castellana et al. (2007) proposed a new approach to performing change detection analyses based on a combination of supervised and unsupervised techniques is presented. Experimental results are based on image data. Wang et al. (2010) examined an unsupervised search method to discover motifs from multivariate time series data. The algorithm first scans the entire series to construct a list of candidate motifs in linear time, the list is then used to populate a sparse self-similarity matrix for further processing to generate the final selections. In contrast, the algorithm to be proposed is based on time-stamped data.

3.3 Notches in sales series

The change in sales of an item could be defined by the change of its support over time. The support of an item results in a *support history* (Bottcher et al., 2008) of the item in time databases. An analysis of a support history could be important in understanding customers' behavior (Adhikari et al., 2009). While dealing with the support history, the size of a database is an important issue. Support 0.129 from a database containing 1,000 transactions might be less than the support 0.091 from a database containing 1,00,000 transactions. Thus, a mere analysis of the support history over time might not be effective in an application. One needs to analyze the supports along with the sizes in time databases. In Example 1, we observe that the support of *A* has been decreased from year 1970 to 1971. But, the sales of *A* has been increased from the year 1970 to 1971. Hence, a negative change in support of an item might imply a positive change in frequency of the item. Thus, one needs to be careful in dealing with support history of an item in different databases.

Let us consider a company that has been operating for the last k years. For the purpose of studying temporal patterns, yearly databases could be constructed based on a time-stamp. Each of these databases corresponds to a specific period of time. Let D be the collection of customer transactions over k years. For the purpose of defining a temporal pattern we divide D into k yearly databases. Let DT_i be the database corresponding to the i-th year, $i = 1, 2, \ldots, k$. Each of these time databases is mined using a traditional data mining technique (Agrawal & Srikant, 1994; Han et al., 2000). Mining time-stamped databases could help business make better decisions by listening to their suppliers and / or customers via their transactions collected over time (Leonard & Wolfe, 2005).

Over the years, an item may exhibit many consecutive data points having similar sales. As opposed to similar data patterns considering each data point, a limited yet meaningful number of points may play a dominant role in many decision making problems. These meaningful data points could be defined in various ways, like average, peak, or slope of lines (Pratt & Fink, 2002). In the context of the proposed problem, such compression of data points seems to be irrelevant. Given the sales series of an item, one might be interested in identifying abrupt changes in the sales series. The goal of this chapter is to define a new type of pattern based on abrupt variation of sales of an item over the years and to design an algorithm to mine such patterns in time databases.

Over the years, there may exist many ups and downs in sales of an item. One might be interested in identifying abrupt changes of sales in different years. It might be helpful to figure out the causes behind it and to take actions accordingly. Let s_i (A) be the sales of item A for the i-th year, $i = 1, 2,\ldots, k$. We define the change in sales series of item A at year i as follows (Singh & Stuart, 1998).

The change in sales series at year i is *increasing* if

$$s_{i-1}(A) < s_i(A) < s_{i+1}(A),\ i = 2, 3,\ldots, k\text{-}1 \qquad \ldots(1)$$

The change of sales series at year i is *decreasing* if

$$s_{i-1}(A) > s_i(A) > s_{i+1}(A),\ i = 2, 3,\ldots, k\text{-}1. \qquad \ldots (2)$$

$$[\, s_{i-1}(A) < s_i(A) \text{ and } s_i(A) > s_{i+1}(A)] \text{ or } [s_{i-1}(A) > s_i(A) \text{ and } s_i(A) < s_{i+1}(A)],\ i = 2, 3,\ldots, k\text{-}1. \qquad \ldots(3)$$

The notion of *strict extrema* (Fink & Gandhi, 2007) at a year corresponding to an item is defined as follows.

Let $s_i(A)$ be the amount of sales of an item A at year i, $i = 1, 2,\ldots, k$. There exists a *strict extrema* at year i for the item A if the change of support history of A at year i is altering. Based on the concept of strict extrema, we define a notch as follows, for the first time.

Definition 1. There exists a *notch* at year i for the item A if there is a strict extreme at year i in the sales series of item A.

Let $\Delta s_i(A)$ be the difference in sales of item A between the years i and i-1. Now we propose a few definitions as follows.

Definition 2. Let there exist a notch at year i for item A. The notch at year i for item A is *downward* if $\Delta s_i(A) < 0$ and $\Delta s_{i+1}(A) > 0$.

Definition 3. Let there exists a notch at year i for item A. The notch at year i for item A is *upward* if $\Delta s_i(A) > 0$ and $\Delta s_{i+1}(A) < 0$.

Itemset (Agrawal et al., 1993) could be considered of as a basic type of pattern present in a transactional database. Many important as well as interesting patterns are based on itemset patterns. Similarly an upward / a downward notch could be considered as a basic type of pattern in time databases. In Section 4, we illustrate how the notion of notch could help analyzing a special type of trend in time databases. Thus, it is important to mine notches in time databases. One could scan the sales series of an item to identify its interesting notches. Let n and k be the number of items in time databases and the number of time-stamped (yearly) databases, respectively. One could simply scan the sales series of an item to identify its interesting notches. For each item there are k sales data. Then the time complexity of identifying notches is $O(n \times k)$.

3.4 Generalized notch

Based on the concept of notch, we present here the notion of a generalized notch in time databases. Let us refer to Figure 3.1. There are two downward notches in the years 1971 and 1975 having sales 854 and 537, respectively. The concept of notch can be generalized based on strict extrema as mentioned in Section 3.3. One could notice in Figure 3.1 that the downward notch in the year 1975 is wider than that of 1971. The width of a downward generalized notch is based on the two consecutive local maxima within which the downward generalized notch is enclosed. The width of the downward generalized notch in 1975 is 1978 – 1972 = 6. Similarly, the width of an upward generalized notch is based on the two consecutive local minimums within which the upward generalized notch is enclosed. The width of the upward generalized notch in 1972 is 1975 – 1971 = 4. Based on the above discussion, we define width of a generalized notch as follows.

Definition 4. Let there be a generalized downward (upward) notch in the year i. Also, let the generalized downward (upward) notch be enclosed with the local maximums (minimums) in the years i_1 and i_2 ($> i_1$). The width of the generalized notch in the year i is equal to $i_2 - i_1$.

The width of a generalized notch could be divided into left width and right width. The left width and right width are equal to $(i - i_1)$ and $(i_2 - i)$, respectively. In case of downward generalized notch the sales value gradually decreases, and then attains the minimum value, and then it gradually increases. Thus, the change of sales value between two consecutive years seems to be an important characteristic of a generalized notch. In this regard, one might be interested in the change of sales for a year as compared to its previous year. Also, the sales at year i as compare to sales of year i_1 and i_2 are important characteristics of a generalized notch.

Accordingly, one could define left-height and right-height of the generalized notch in the year i for an item A as follow: *left-height* $(A, i) = |sales(A, i_1) - sales(A, i)|$, and *right-height* $(A, i) = |sales(A, i) - sales(A, i_2)|$. We define the height of a generalized notch as follows.

Definition 5. Let there exists a generalized downward (upward) notch in the year i. Also, let the generalized downward (upward) notch be enclosed with the local maximums (minimums) i_1 and i_2 ($> i_1$). The height of the generalized notch in the year i is equal to *maximum* {*left-height* (A, i), *right-height* (A, i)}.

Based on the concept of generalized notch, we focus on the notion of iceberg.

3.5 Iceberg notch

An analysis of sales series of items is an important issue. In view of performing this task, one could analyze the sales series for each item. In analyzing a sales series in-depth, it is evident that an existence of a notch might be an indication of a bigger notch. This represents an exceptionality of sales of an item. Based on such an exception, we define iceberg in time databases as follows.

Definition 6. An *iceberg* notch is a generalized notch that satisfies the following conditions: (i) The height of the generalized notch is greater than or equal to α, and (ii) The width of the generalized notch is greater than or equal to β. Both α and β are user-defined thresholds.

An iceberg notch is a generalized notch having a larger height and a larger width than α and β. The concept of iceberg in data management is not new. For example, iceberg queries (Han & Kamber, 2001) are commonly used in data mining, particularly in market basket analysis.

Let us illustrate the concept of an iceberg using an example. Let the value of α and β be set to 300 and 5, respectively. Also, let the values of i, i_1 and i_2 be 1975, 1972, and 1978, respectively (with respect to Figure 1). We observe that *leftHeight*(*A*, 1975) = |*sales*(*A*, 1972) – *sales*(*A*, 1975)| = |905 – 537| = 368 and *rightHeight*(*A*, 1975) = |*sales*(*A*, 1975) – *sales*(*A*, 1978)| = |537 – 820| = 283, respectively. Therefore, the height of the iceberg is *maximum* {368, 283} i.e. $368 \geq \alpha$. Also, $i_2 - i_1 = (1978 - 1972) = 6 \geq \beta$. So, there exists an interesting downward iceberg notch in the year 1975. We also observe an upward notch in the year 1972 with height $368 \geq \alpha$ and width $(1975 - 1971) = 4 < \beta$. So, the upward notch in the year 1972 is not an iceberg.

3.6 Sales series

A sales series of an item might provide an interesting information about the item. It is basically the same as the support history of the item. As noted above, in many problems, it is preferable to analyze sales series rather than looking at the support history of an item. Many temporal patterns might originate by analyzing such types of temporal series.

Each data in a sales series can be mapped into a member in the set {–1, 0, 1} by comparing with the previous data in the same series. Thus, a time-stamped series could be mapped into a ternary series. It provides a simplified view of the original time-stamped series data. Such simplified view might provide some useful information. The procedure for mapping a time-stamped series into a ternary series is illustrated in the following example.

Example 2. Consider the sales data given in Example 3.1. The sales of item *A* in 1971 decreased from the sales in 1970. We note this behavior by putting –1 in the ternary series of item *A* corresponding to year 1971.

The sales of item *A* in 1972 increased over the sales in 1971. We note this behaviour by putting +1 in the ternary series of item *A* corresponding to year 1972. If the sales of item *A* in any year remains same as that of previous year then we note this behaviour by putting 0 in the ternary series. Thus, we obtain the ternary series (*TS*) of item *A* in the following form:

Year	1970	1971	1972	1973	1974	1975	1976	1977	1978	1979
TS(*A*)		–1	+1	–1	–1	–1	+1	+1	+1	–1

In the above series, one can observe the existence of a generalized notch. The width of a downward generalized notch can be obtained from a run of –1's and the subsequent run of +1's. The width is obtained by adding the number of –1's in the first run and the number of +1's in the second run. A similar procedure can be followed for finding width of an upward generalized notch. In *TS*(*A*) we observe a downward generalized notch in 1975. The width of this notch is equal to 3 + 3 = 6. Also, there exists an upward generalized notch in 1978 having width of 4.

A slightly different procedure could also be followed for obtaining a ternary series corresponding to a sales series of an item. Let x and y be the sales for the year 1970 and 1971, respectively. Let δ be the level of significance of difference in sales. We put +1 in the ternary series of the item corresponding to year 1971, if $y - x > \delta$. We put –1 in the ternary series of the item corresponding to year 1971, if $x - y > \delta$. We put 0 in the ternary series of the item corresponding to year 1971, if $|x - y| \leq \delta$. The method of obtaining a ternary series using this procedure might be useful in many situations, since a small change in sales value might be insignificant in many situations. This procedure is more useful than the previous one.

3.7 Mining icebergs in time-stamped databases

Let there are n items in time databases. For each item in time databases there exists a time-stamped series containing k data. In this section we are interested in identifying icebergs in each time-stamped series.

For mining icebergs in time databases, we make use of an existing frequent itemset mining algorithm (Agrawal & Srikant, 1994; Han et al., 2000). For the requirement of proposed problem one needs to mine the frequencies of each item in the time databases. Based on the discussion held in previous sections, we design an algorithm for mining icebergs in time databases. Let n and k be the number of items in time databases and the number of time databases, respectively. In this algorithm, we use a two-dimensional array F for storing frequencies of all items in time databases. F consists of n rows and $k + 1$ columns. The first column contains the items in time databases. For example, $F(i)(1)$ contains the i-th item in time databases, $i = 1, 2, \ldots, n$. The i-th row of F contains i-th item and its frequencies in k time databases. For example, $F(i)(j)$ contains the frequency of i-th item in $(j - 1)$-th database, $j = 2, 3, \ldots, k+1$. Therefore, we need to check the existence of a generalized notch using the values in the columns from 2 to $k + 1$.

For the purpose of computing interestingness of a generalized notch, we determine the change of sales of a local minimum (maximum) with respect to its previous and next local maximum (minimum). During the process of mining icebergs, the generalized notches are kept in array *GN*. A generalized notch can be described by the following attributes: left year (*leftYear*), right year (*rightYear*), item (*item*), year of occurrence (*year*), type of generalized notch (*type*), change of

sales at the year of occurrence with respect to the previous local extremum (*leftHeight*), change of sales at the year of occurrence with respect to the next local extremum (*rightHeight*), width of generalized notch (*width*) and the sales at the year of occurrence (*sales*). The goal of the proposed algorithm is to find all the interesting icebergs for each item in time databases. The algorithm is given as follows.

Algorithm 1. Mine icebergs in time-stamped databases.

procedure *MineIcebergs* (*k*, *F*, α, β)

Inputs: *k*, *F*, α, β

k: number of yearly databases

F: array of frequencies of items in yearly databases

α: user-defined threshold of height of a generalized notch

β: user-defined threshold of width of a generalized notch

Outputs:

Interesting icebergs in time databases

01: **let** *index* = 1;

02: **for** *i* = 1 to *n* **do**

03: **let** *j* = 2; **let** *left* = 2; **let** *flat* = false; **let** *prevDown* = false; **let** *prevUp* = false;

04: **while not** end of the sales series corresponding to *i*-th item **do**

05: **if** there is a downward trend **and** *prevUp* is false **then**

06: find *mid*, *leftWidth*; **let** *prevDown* = true;

07: compute *leftHeight*; **go to** 04;

08: **end if** {05}

09: **if** there is an upward trend **and** *prevDown* is true **then**

10: find *left, mid, right, leftWidth, rightWidth*; **let** *prevDown* = false;

11: **let** *leftHeight* = *rightHeight*;

12: compute *rightHeight*;

13: *GN*(*index*).*type* = *down*; **go to** 30;

14: **end if** {09}

15: **if** there is an upward trend **and** *prevDown* is false **then**

16: **let** *prevUp* = true; find *mid, leftWidth*;

17: compute *leftHeight*; **go to** 04;

18: **end if** {15}

19: **if** there is a downward trend **and** *prevUp* is true **then**

20: find *left, mid, right, leftWidth, rightWidth*;

21: **let** *prevUp* = false; **let** *prevDown* = true;

22: **let** *leftHeight* = *rightHeight*;

23: compute *rightHeight*;

24: *GN*(*index*).*type* = *up*; **go to** 30;

25: **end if** {19}

```
26:      if the sales of the j-th and (j+1)-th year remain same then
27:        find left; let flat = true; let prevDown = false; let prevUp = false;
28:        go to 04;
29:      end if
30:      if flat is false then
31:        compute height as defined in Definition 5;
32:        if the current generalized notch satisfies the criteria α and β then
33:          store it in GN(index); increase index by 1;
34:        end if
35:        if the current generalized notch is downward then
36:            let prevDown = false; let prevUp = true;
37:        else if the current generalized notch is upward then
38:              let prevDown = true; let prevUp = false;
39:            end if
40:        end if {35}
41:      end if {30}
42:    end while {04}
43:  end for {02}
44:  display icebergs from GN;
end procedure
```

The lines 2-43 are repeated for each item in time databases. In each repetition, the interesting icebergs corresponding to an item are identified. The variable *index* is used to index array *GN*. The variable *j* is used to keep track of current sales data of the item under consideration. The starting value of *j* is 2, since the sales data for the first year of an item is kept staring from column number 2 of array *F*. We use three Boolean variables viz., *flat*, *prevUp* and *prevDown*. While identifying downward (or, upward) generalized notches, we first go through its left leg of a generalized notch. After reaching its minimum / maximum value, if the next point also attains the same value then *flat* becomes true. The width of a generalized notch is determined by the following years: left year (*left*), middle year (*mid*), and right year (*right*). Accordingly, the width of a generalized notch has two components: left width (*leftWidth*) and right width (*rightWidth*). After identifying the left leg of a downward (or, upward) generalized notch, *prevDown* (or, *prevUp*) becomes true. After storing the details of the current generalized notch *index* gets increased by 1 (line 33). We identify generalized notches for each item in time databases. For this purpose, we introduce a for-loop at line 02 which ends at line 43. Some lines, e.g. lines 40, 41, 42, 43, are ended with a number enclosed in curly brackets, to mark the ends of composite statement starting with the line number kept in curly bracket.

Lines 2 and 4 repeat for n and k times respectively. In other words, the sales series corresponding to each item is processed for identifying icebergs. Thus, the time complexity of lines 1-43 is $O(n \times k)$. Again, the time complexity of line 44 can not be more than $O(n \times k)$, since the number of interesting icebergs is always less than $n \times k$.

Theorem 1. Corectness of the *MineIcebergs* algorithm.

Proof: Consider that there are *n* items in *k* time-stamped databases. Each sales series is processed using lines 2-43. For the purpose of mining interesting icebergs, each sales series is checked completely by applying a while-loop shown in lines 4-42. A sales series can start with one of the following three ways: (a) showing a downward trend, (b) showing an upward trend (c) remained at a constant level. The algorithm handles each of these cases separately.

Case (a) has been checked at the line numbered as 05. Once a downward trend changes we again go back to while-loop at line 04 for finding one of the following two possibilities: an upward trend and a constant sales.

Case (b) has been checked at the line number 15. Once the upward trend changes we again go back to while-loop at line 04 for finding one of the following two possibilities: a downward trend and a constant sales.

Case (c) has been checked at the line number 26. Once the flatness changes we again go back to while-loop at line 04 for finding one of the following two possibilities: a downward trend and an upward trend.

Once the left leg of a downward generalized notch is detected in lines 5-8, its right leg is detected in lines 9-14. When the left leg of an upward generalized notch is detected in lines 15-18, its right leg is detected in lines 19-25. After detecting a generalized notch at lines 13 and 24, we go to line 30 for detecting its interestingness and re-initializing required Boolean variables. Thus, the above algorithm considers all the possibilities that would arise in each sales series.

3.8 Experimental studies

We have carried out several experiments for mining generalized notches in different databases. All the experiments are performed on a 1.6 GHz Pentium IV with 256 MB of memory using visual C++ (version 6.0) software. We present experimental results using four real databases and two synthetic databases. The databases *mushroom*, *retail* (Frequent itemset mining dataset repository) *ecoli* and *BMS-WebView-1*are real-world databases. The real databases *BMS-Web-Wiew-1* can be found from KDD CUP 2000 (Frequent itemset mining dataset repository). Database *ecoli* is a subset of *ecoli database* (UCI ML repository). The synthetic dataset *T10I4D100K* was generated using the generator from the IBM Almaden Quest research group. *Random-68* is also a synthetic database and has been generated for the purpose of conducting experiments. The characteristics of these databases are given in Table 3.1. The density, i.e. *ALT*/*NI*, of *mushroom*, *ecoli*, *random-68*, *retail*, *BMS-WebView-1*, and *T10I4D100K* are 0.2, 0.07, 0.08, 0.00113, 0.0003, and 0.013, respectively.

Table 3.1 Database characteristics

Database	*NT*	*ALT*	*AFI*	*NI*
mushroom (*M*)	8124	24.000	1624.800	120
ecoli (*E*)	336	7.000	25.835	91
random-68 (*R*)	3000	5.460	280.985	68
retail (*Rt*)	88,162	11.306	99.674	10,000
BMS-WebView-1	1,49,639	2.000	44.575	6714
T10I4D100K	1,00,000	11.102	1276.124	870

The symbols used in Tables 3.1 and 3.2 have following meaning: *D*, *NT*, *ALT*, *AFI*, and *NI*

denote database, the number of transactions, average length of a transaction, average frequency of an item, and number of items, respectively.

The databases *mushroom*, *ecoli*, *random-68* and *retail* have been divided into 10 sub-databases, called yearly databases, for the purpose of conducting experiments. The databases *BMS-WebView*-1 and *T10I4D100K* have been divided into 20 databases. The databases obtained from *mushroom*, *ecoli*, *random-68* and *retail* are named as M_i, E_i, R_i, and Rt_i, $i = 0, 1, \ldots, 9$. The databases obtained from *BMS-WebView*-1 and *T10I4D100K* are named as B_i and T_i, $i = 0, 1, \ldots, 19$. We present some characteristics of the input databases in Table 3.2.

In Table 3.3, 3.4, 3.5 and 3.6 we have represented upward generalized notch as '*u*' and downward generalized notch as '*d*'. In *mushroom*, there are many items having high frequency as it is relatively dense. Also, we get many generalized notches having relatively large height as shown in Table 3.3. On the other hand, the items in *retail* is somewhat skewed in the sense that some generalized notches for few items have large height. But, the items in *random-68* and *ecoli* have got more or less uniform distribution. Many generalized notches in these two databases have similar height. Unlike *mushroom* and *retail*, *ecoli* and *random-68* are smaller in size and contain items with lesser variations. In these two databases the maximum heights are 13 and 30, respectively. With respect to width of a generalized notch, we have got similar characteristics. In *mushroom* and *retail* the generalized notches are wider than that of other two databases. These facts are quite natural, since *mushroom* and *retail* are bigger than *random-68* and *ecoli*. The variation of sales over the years for an item in *mushroom* and *retail* is higher. Also, we observe that many upward generalized notch is followed by a downward generalized notch and vice versa.

This is because of the fact that two consecutive different types (a ‘u’ type followed by a ‘d’ type or a ‘d’ type followed by an ‘u’ type) generalized notches share a common leg. For example, a ‘u’ type generalized notch at year 4 is followed by a ‘d’ type generalized notch at year 7, for item 116 in *mushroom*. Similarly, a ‘d’ type generalized notch at year 6 is followed by a ‘u’ type generalized notch at year 7, for item 0 in *ecoli*. Also, we observe that some items have both long height and long width. For example, item 56 has height 796 and width 8. These values are significantly high as compare to other items in the time databases. Also, this is true for item 67. In *BMS-WebView*-1 database items 333469 and 110877 have maximum variation. Therefore, only these two items are appearing among top ten generalized notches and their heights vary from 344 to 506. Similarly, items 966, 998 and 419 in *T10I4D100K* share common legs and they have more variations. From Table 3.6 one could conclude that generalized notches are sharper in *BMS-WebView*-1 as compared to *T10I4D100K*. Some more observations are made from Tables 3.3 and 3.5. In *mushroom* items 56 {5(810}, 67{6(103)} and 94{5(22)} satisfy user-defined threshold at α = 300 and β = 4. So, these items appear in both the tables. Since *AFI* of *mushroom* is more, the height and width of generalized notches are very large as compare to thresholds. Similarly, four generalized notches for items 42, 35, and 41are common for *ecoli* database. In *random-68* only item 18 satisfies both the thresholds. From Tables 3.4 and 3.6 one could notice that *retail* does not have any common item and items 41, 0 and 48 having more variation in sales. Like *retail*, in *BMS-WebView-1*and *T10I4D100K* different items appear in both the tables. From the experimental results we could conclude that items having maximum variations over many consecutive years will appear in both the tables. The highest width of generalized notch (10) appears for *BMS-WebView-1*and lowest (4) appears in *ecoli*.

Table 3.2 Characteristics of time databases

D	*NT*	*ALT*	*AFI*	*NI*	*D*	*NT*	*ALT*	*AFI*	*NI*
M_0	812	24.000	295.273	66	M_5	812	24.000	221.454	88
M_1	812	24.000	286.588	68	M_6	812	24.000	216.533	90
M_2	812	24.000	249.846	78	M_7	812	24.000	191.059	102
M_3	812	24.000	282.435	69	M_8	812	24.000	229.271	85
M_4	812	24.000	259.840	75	M_9	816	24.000	227.721	86
E_0	33	7.000	4.620	50	E_5	33	7.000	3.915	59
E_1	33	7.000	5.133	45	E_6	33	7.000	3.500	66
E_2	33	7.000	5.500	42	E_7	33	7.000	3.915	59
E_3	33	7.000	4.812	48	E_8	33	7.000	3.397	68
E_4	33	7.000	3.397	68	E_9	39	7.000	4.550	60
R_0	300	5.590	28.677	68	R_5	300	5.140	26.677	68
R_1	300	5.417	28.000	68	R_6	300	5.510	28.353	68
R_2	300	5.360	27.647	68	R_7	300	5.497	28.338	68
R_3	300	5.543	28.456	68	R_8	300	5.537	28.471	68
R_4	300	5.533	28.382	68	R_9	300	5.477	28.235	68
Rt_0	9000	11.244	12.070	8384	Rt_5	9000	10.856	16.710	5847
Rt_1	9000	11.209	12.265	8225	Rt_6	9000	11.200	17.416	5788
Rt_2	9000	11.337	14.597	6990	Rt_7	9000	11.155	17.346	5788
Rt_3	9000	11.490	16.663	6206	Rt_8	9000	11.997	18.690	5777
Rt_4	9000	10.957	16.039	6148	Rt_9	7162	11.692	15.348	5456
B_0	7482	2.000	5.016	2983	B_{10}	7482	2.000	4.573	3272
B_1	7482	2.000	4.494	3330	B_{11}	7482	2.000	4.895	3057
B_2	7482	2.000	5.782	2588	B_{12}	7482	2.000	4.636	3228
B_3	7482	2.000	4.359	3433	B_{13}	7482	2.000	4.805	3114
B_4	7482	2.000	4.228	3539	B_{14}	7482	2.000	4.192	3570
B_5	7482	2.000	4.194	3568	B_{15}	7482	2.000	4.656	3214
B_6	7482	2.000	3.786	3952	B_{16}	7482	2.000	5.379	2782
B_7	7482	2.000	3.477	4304	B_{17}	7482	2.000	4.863	3077
B_8	7482	2.000	4.168	3590	B_{18}	7482	2.000	4.654	3215
B_9	7482	2.000	4.365	3428	B_{19}	7481	2.000	4.953	3021
T_0	5000	11.123	64.968	856	T_{10}	5000	11.113	64.913	856
T_1	5000	10.987	63.880	860	T_{11}	5000	11.165	64.988	859
T_2	5000	11.189	65.128	859	T_{12}	5000	11.127	64.617	861
T_3	5000	11.078	64.330	861	T_{13}	5000	11.089	64.694	857
T_4	5000	11.003	63.895	861	T_{14}	5000	11.169	65.088	858
T_5	5000	11.131	64.867	858	T_{15}	5000	11.028	64.338	857
T_6	5000	11.171	64.645	864	T_{16}	5000	11.132	64.795	859
T_7	5000	11.075	64.764	855	T_{17}	5000	11.031	64.661	853
T_8	5000	11.123	65.121	854	T_{18}	5000	11.072	64.374	860
T_9	5000	11.151	64.755	861	T_{19}	5000	11.090	64.856	855

Table 3.3 Top 10 generalized notches in *M*, *E* and *R* databases (according to height)

$M(\alpha)$ at $\beta = 4$				$E(\alpha)$ at $\beta = 2$				$R(\alpha)$ at $\beta = 3$			
item	*year* (sales)	*type*	*height*	*item*	*year* (sales)	*type*	*height*	*item*	*year* (sales)	*type*	*height*
116	4(1489)	*u*	1344	42	2(14)	*u*	13	48	4(48)	*u*	30
116	7(353)	*d*	1136	42	6(1)	*d*	13	48	5(18)	*d*	30
114	2(1131)	*u*	1062	35	6(0)	*d*	12	27	5(19)	*d*	26
114	4(69)	*d*	1062	0	6(10)	*d*	10	27	8(45)	*u*	26
56	5(810)	*u*	796	0	7(20)	*u*	10	36	6(39)	*u*	26
56	9(14)	*d*	796	39	4(11)	*u*	10	36	8(13)	*d*	26
67	6(103)	*d*	704	39	6(1)	*d*	10	18	8(21)	*d*	25
94	5(2)	*d*	650	41	3(11)	*u*	10	3	5(41)	*u*	24
94	9(652)	*u*	650	41	6(1)	*d*	10	3	7(17)	*d*	24
1	3(69)	*d*	644	34	2(2)	*d*	9	36	2(41)	*u*	23

Table 3.4 Top 10 generalized notches in *Rt*, *B* and *T* databases (according to height)

$Rt(\alpha)$ at $\beta = 2$				$B(\alpha)$ at $\beta = 3$				$T(\alpha)$ at $\beta = 2$			
item	*year* (sales)	*type*	*height*	*item*	*year* (sales)	*type*	*height*	*item*	*year* (sales)	*type*	*height*
41	4(2617)	*u*	2617	333469	9(582)	*u*	506	966	4(297)	*d*	122
41	9(2355)	*u*	2355	333469	12(76)	*d*	506	966	6(419)	*u*	122
0	2(2331)	*d*	1315	333469	4(151)	*d*	388	998	2(346)	*u*	103
48	9(4544)	*u*	932	333469	6(539)	*u*	388	998	6(243)	*d*	103
48	4(4704)	*u*	711	333449	9(474)	*u*	379	966	8(395)	*u*	93
0	3(2403)	*u*	609	333449	11(95)	*d*	379	966	10(302)	*d*	93
0	4(1794)	*d*	609	333449	4(148)	*d*	372	829	16(431)	*u*	90
0	7(1619)	*u*	571	333449	6(520)	*u*	372	966	11(389)	*u*	87
8978	2(556)	*u*	556	110877	5(353)	*u*	344	419	4(179)	*d*	85
0	5(2089)	*u*	512	110877	10(9)	*d*	344	419	6(264)	*u*	85

Table 3.5 Top 10 generalized notches in different databases (according to width) at a given α

M (β) at $\alpha = 300$				E (β) at $\alpha = 2$				R (β) at $\alpha = 5$			
item	*year* (sales)	*type*	*width*	*item*	*year* (sales)	*type*	*width*	*item*	*year* (sales)	*type*	*width*
56	5(810)	*u*	8	35	6(0)	*d*	6	1	5(19)	*d*	6
11	4(427)	*u*	7	42	6(1)	*d*	6	11	8(21)	*d*	6
13	6(39)	*d*	7	33	7(1)	*d*	5	43	5(36)	*u*	6
16	5(361)	*u*	7	41	3(11)	*u*	5	6	3(32)	*u*	5
67	6(103)	*d*	7	42	2(14)	*u*	5	11	4(32)	*u*	5
94	5(2)	*d*	7	49	6(1)	*d*	5	18	8(21)	*d*	5
98	2(404)	*u*	7	52	4(1)	*d*	5	47	5(31)	*u*	5
11	8(32)	*d*	6	40	4(8)	*u*	4	50	4(18)	*d*	5
52	6(703)	*u*	6	45	2(8)	*u*	4	50	8(35)	*u*	5
53	6(109)	*d*	6	45	5(1)	*d*	4	53	6(40)	*u*	5

Table 3.6 Top 10 generalized notches in different databases (according to width) at a given α

Rt (β) at $\alpha = 20$				B (β) at $\alpha = 50$				T (β) at $\alpha = 5$			
item	*year* (sales)	*type*	*width*	*item*	*year* (sales)	*type*	*width*	*item*	*year* (sales)	*type*	*width*
9823	7(30)	*u*	9	112551	9(6)	*d*	10	673	8(71)	*d*	8
2046	4(133)	*u*	8	335213	10(4)	*d*	10	651	11(82)	*u*	8
3321	7(24)	*u*	8	112339	5(224)	*u*	9	524	4(29)	*u*	8
411	4(32)	*u*	8	335185	4(130)	*u*	9	283	15(229)	*u*	7
3121	4(0)	*d*	8	112407	5(107)	*u*	9	487	15(139)	*d*	7
2919	6(32)	*u*	8	335181	5(54)	*u*	9	336	13(42)	*d*	7
103	7(267)	*u*	7	335213	5(54)	*u*	9	523	11(117)	*u*	7
855	3(118)	*u*	7	335177	5(51)	*u*	9	658	14(88)	*d*	7
1659	3(116)	*u*	7	110877	5(353)	*u*	8	733	16(63)	*u*	7
976	6(55)	*d*	7	110315	11(310)	*u*	8	807	10(29)	*u*	7

We have also noticed that from the above Tables higher frequency in sales indicate upward notch otherwise downward.

We have also reported execution time with respect to the number of data sources. We observe in Figures 3.2, 3.3, 3.4, 3.5, 3.6 and 3.7 that the execution time increases linearly as the number of databases increases. The size of each input database generated from *mushroom*, *retail*, *BMS-WebView-1* and *T10I4D100K* are significantly larger than that of *ecoli* and *random-68*. The density of these two databases are also same. As a result we observe similar type of graphs in Figures 3.3 and 3.4. We have fixed α and β for *ecoli* and *random-68* at lower level, since variation of frequencies of an item is lesser. In both the cases we have observed execution time increases linearly with the increase of number of databases.

We observe that the execution time of *retail* (Figure 3.5) is significantly larger than other databases, since each of the time databases is comparatively larger and the number of items are highest among all datasets. In Figure 3.6 and 3.7 we have considered same α and β for *BMS-WebView-1* and *T10I4D100K* respectively. But execution time of *BMS-WebView-1* is significantly larger than the one reported for *T10I4D100K* as the number of items more than that of *T10I4D100K*.

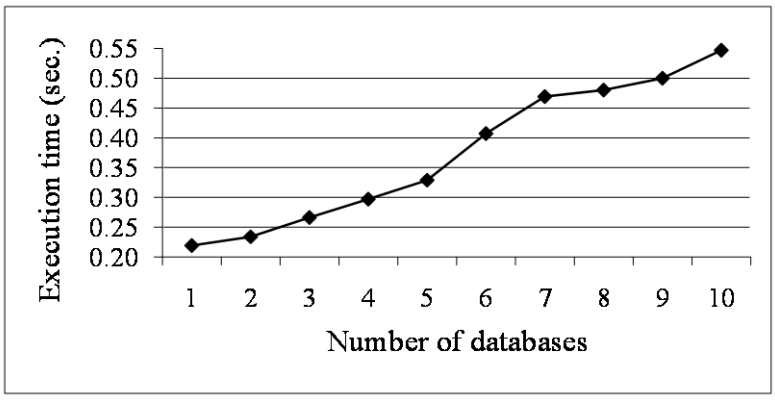

Figure 3.2 Execution time versus the number of databases (*mushroom* at $\alpha = 50$, $\beta = 3$)

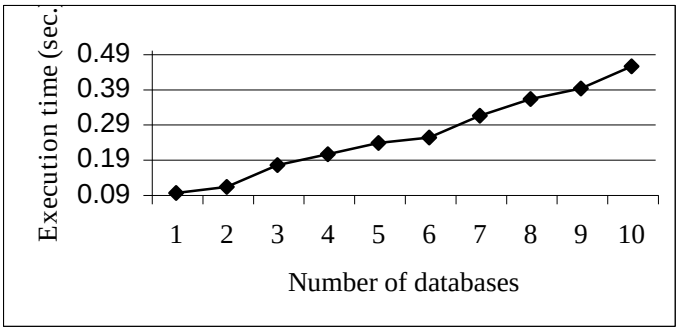

Figure 3.3 Execution time versus the number of databases (*ecoli* at $\alpha = 3$, $\beta = 2$)

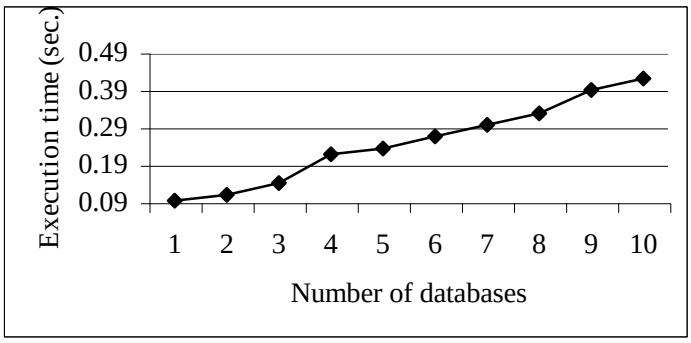

Figure 3.4 Execution time versus the number of databases (*random-68* at $\alpha = 3$, $\beta = 2$)

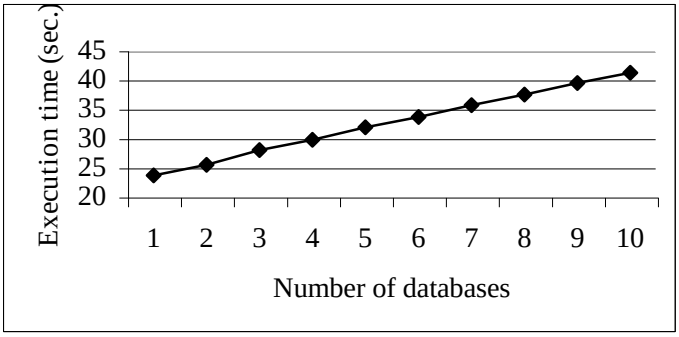

Figure 3.5 Execution time versus the number of databases (*retail* at $\alpha = 20$, $\beta = 2$)

We analysed the nature of graphs by varying user-specified α and β and the number of icebergs in Figures 3.8-3.19.

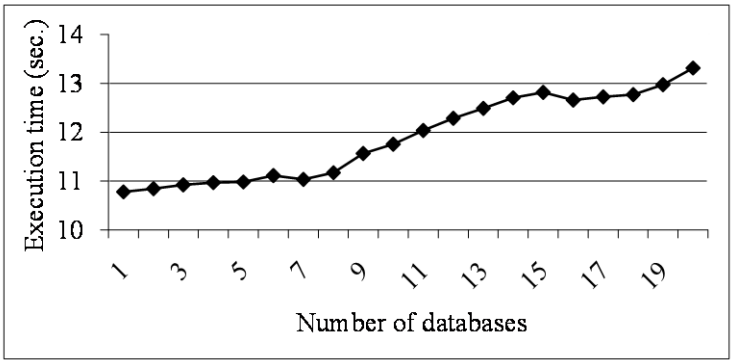

Figure 3.6 Execution time versus the number of databases (*BMS-WebView-1* at $\alpha = 30$, $\beta = 4$)

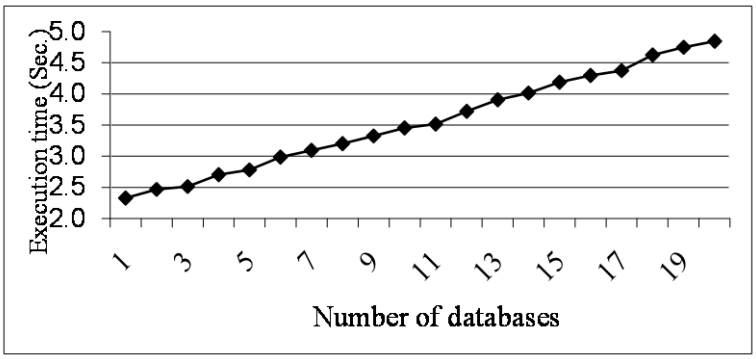

Figure 3.7 Execution time versus the number of databases (*T10I4D100K* at $\alpha = 30$, $\beta = 4$)

In figures it is shown how the number of interesting icebergs decreases with respect to the increase of the values of α and β. In *mushroom*, *ALT* and *AFI* are higher as compared to other databases. Therefore, we start changing the values α proceeding from 100 (Figure 3.8). Initially, in all the cases the number of icebergs decreases significantly. Afterwards, the decrease is not so significant. For higher values of α and β very few icebergs are extracted. Selecting the appropriate α and β are crucial, since lesser α and β generate too many icebergs and vice versa. This means that the time cost for mining icebergs is increased as iceberg increases. Therefore, it is important to specify the appropriate α and β in order to reduce the number of icebergs.

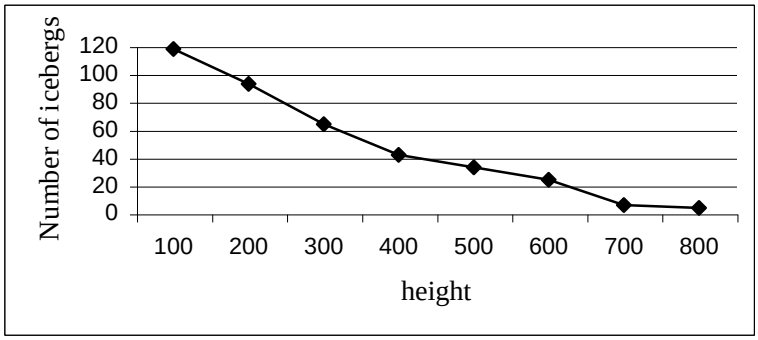

Figure 3.8 Number of interesting icebergs versus height (α) for *mushroom* ($\beta = 3$)

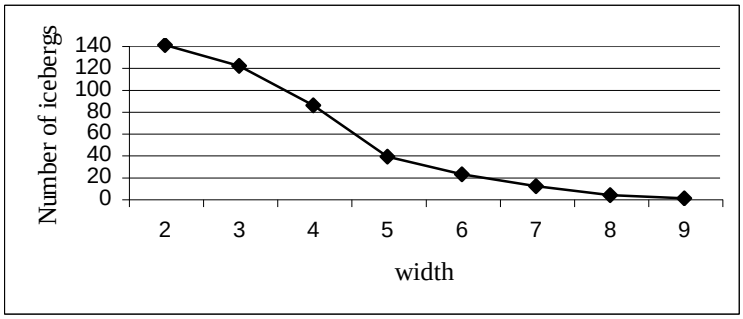

Figure 3.9 Number of interesting icebergs versus width (β) for *mushroom* ($\alpha = 50$)

ALT is smaller for *ecoli* and *random-68*. As a result, the height of an iceberg remains smaller. We start α from 2 and 5 for *ecoli* and *random-68*, respectively (Figures 3.10 and 3.12). We do not obtain any interesting icebergs for the width greater than or equal to 7.

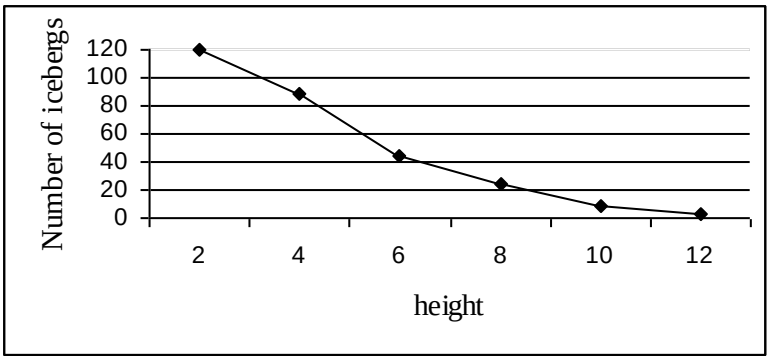

Figure 3.10 Number of interesting icebergs versus height (α) for *ecoli* ($\beta = 2$)

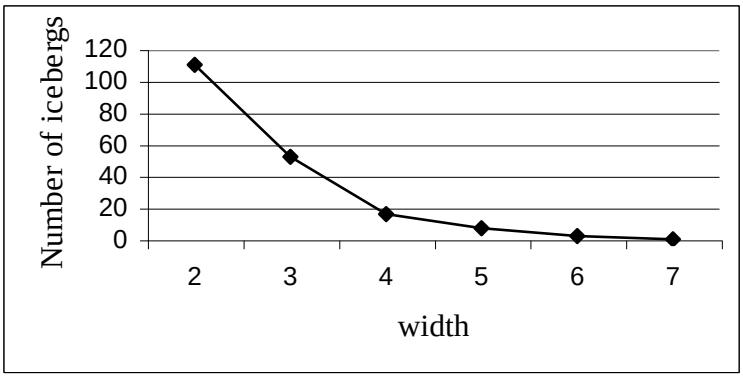

Figure 3.11 Number of interesting icebergs versus width (β) for *ecoli* ($\alpha = 3$)

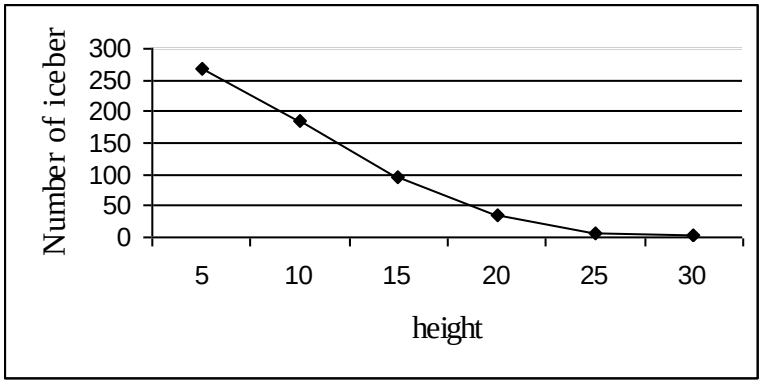

Figure 3.12 Number of interesting icebergs versus height (α) for *random-68* ($\beta = 2$)

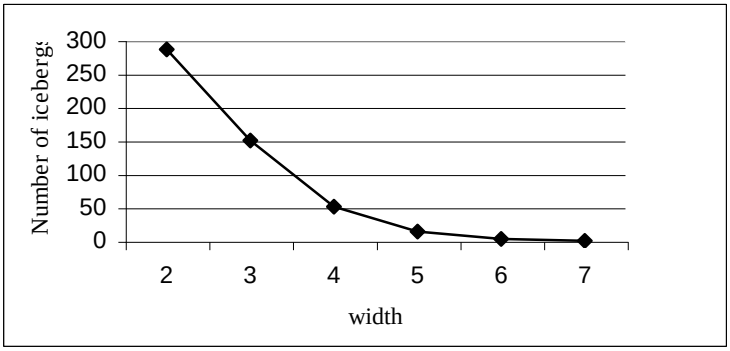

Figure 3.13 Number of interesting icebergs versus width (β) for *random-68* ($\alpha = 3$)

Retail and *BMS-WebView-1* datasets show the similar characteristics for number of interesting icebergs (Figures 3.14, 3.16 and Figures 3.15, 3.17)

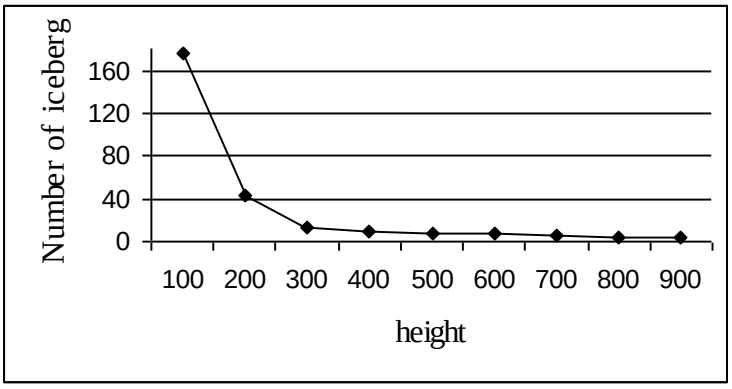

Figure 3.14 Number of interesting icebergs versus height (α) for *retail* ($\beta = 2$)

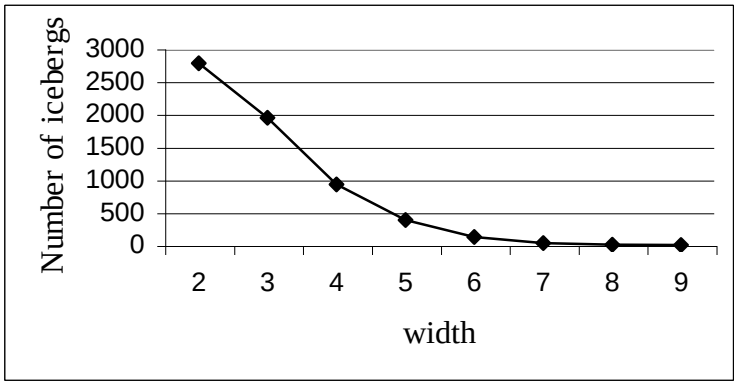

Figure 3.15 Number of interesting icebergs versus width (β) for *retail* ($\alpha = 20$)

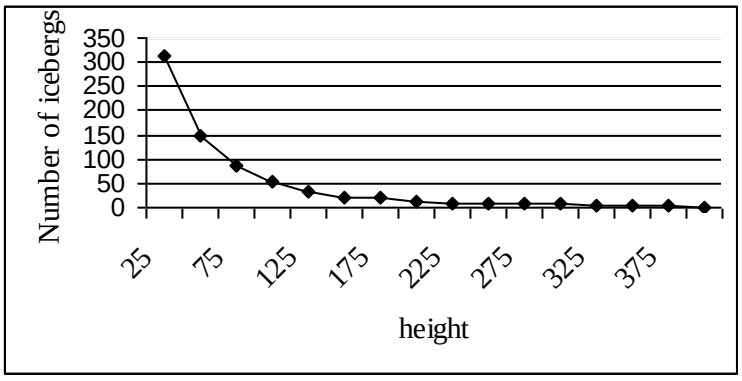

Figure 3.16 Number of interesting icebergs versus height (α) for *BMS-WebView-1* ($\beta = 4$)

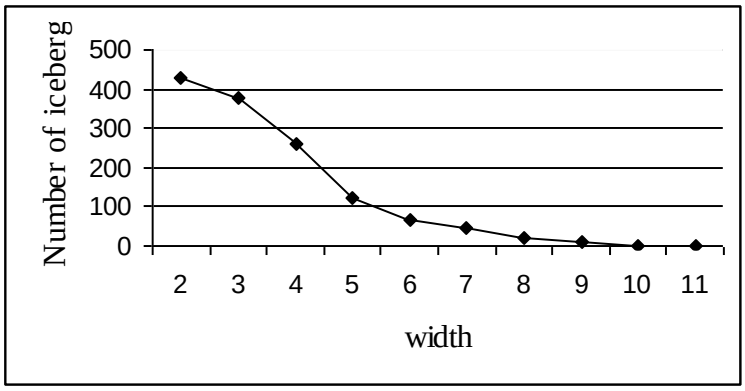

Figure 3.17 Number of interesting icebergs versus width (β) for *BMS-WebView-1* ($\alpha = 30$)

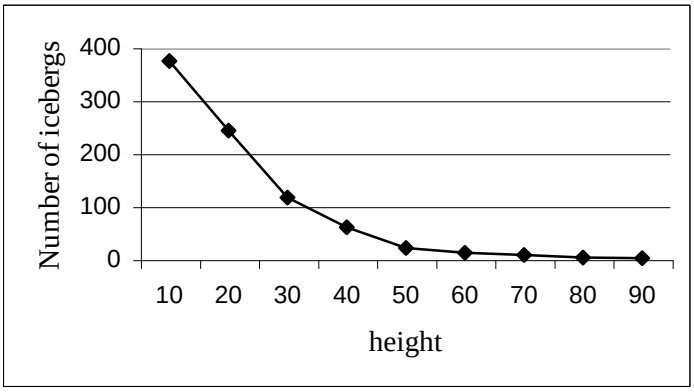

Figure 3.18 Number of interesting icebergs versus height (α) for *T10I4D100K* ($\beta = 5$)

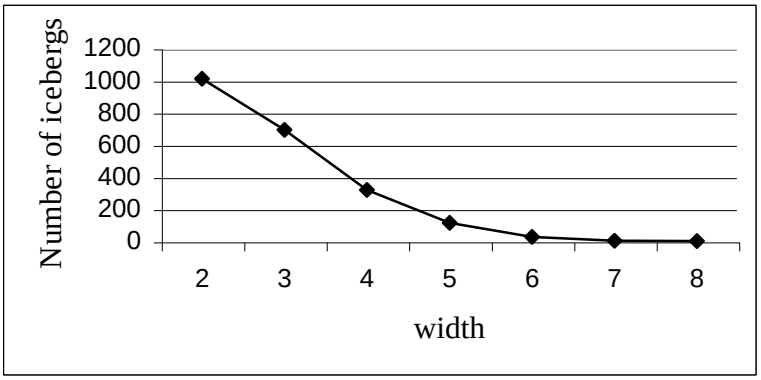

Figure 3.19 Number of interesting icebergs versus width (β) for *T10I4D100K* ($\alpha = 30$)

3.9 Conclusion

The study of temporal patterns in time-stamped databases is an important issue. Many interesting patterns have been discovered in transactional as well as in time series databases. In this chapter, we have proposed definitions of different patterns in time-stamped databases. First, we have introduced the notion of notch in a sales series of an item. Based on this pattern, we have introduced two more patterns viz., generalized notch and iceberg notch, in sales series of an item. Iceberg notch represents a special sales pattern of an item over time. It could be considered as an exceptional pattern in time-stamped databases. Similar to icebergs, some extreme patterns could also be defined from other types of data such as rainfall data and crop-production data.

The investigations of such patterns could be important to understand the purchasing behaviour of customers. They also aid in identifying the reasons for such behavior with the help of domain knowledge. We have designed an algorithm to extract icebergs in time-stamped databases and presented experimental results for real-world and synthetic time-stamped databases.

Chapter 4

Identifying Calendar-based Periodic Patterns

4.1 Introduction

A large amount of data being collected every day has a temporal connotation. For example, databases originate from transactions in a supermarket, logs in a network, transactions in a bank, and events related to manufacturing industry are all inherently related to time. Data mining techniques could also be applied to these databases to discover various temporal patterns to understand the behavior of customers, markets, or monitored processes in different points of time. Temporal data mining is concerned with the analyses of data to find out patterns and regularities from a set of temporal data. In this context sequential association rule (Agrawal & Srikant, 1995), periodical association rule (Li & Deogun, 2005), calendar association rule (Li et al., 2003) calendar-based periodic pattern (Mahanta et al., 2008) and up-to-date pattern (Hong et al., 2009) are some interesting temporal patterns reported in the recent time.

For effective management of business activities, we often wish to discover knowledge from time-stamped data. There are several important aspects of mining time-stamped data including trend analysis, similarity search, forecasting and mining of sequential and periodic patterns. In a database from a retail store, the sales of ice cream in summer and the sales of blanket in winter should be higher than those of the other seasons. Such seasonal behaviour of specific items can only be discovered when a proper window size is chosen for the data mining process (Roddick & Spiliopoulou, 2002). A supermarket manager may discover that turkey and pumpkin pie are frequently sold in together in November in every year. Discovering such patterns may reveal interesting information that can be used for understanding the behaviour of customers, markets or monitored processes in different time periods. However, these types of seasonal patterns cannot be discovered by traditional non-temporal data mining approaches that treat all the data as

one large segment with no attention paid to utilizing the time information of the transactions. If one looks into the entire dataset rather than the transactions that occur in November, it is likely that one will not be able to discover the pattern of turkey and pumpkin pie since the overall support for them will be evidently low. In general, a time-stamped database might exhibit some periodic behaviours. Length of a period might vary from one context to another context. For example, in case of sales of ice cream, the basic time interval could be of three months, since in many regions March, April and May together is considered as summer. Also, in case of sales of blanket, the basic time interval could be considered from November to February in every year. In addition, in many business applications, one might be interested in quarterly patterns over the years, where length of the period is equal to three months. A large amount of data is collected every day in the form of event time sequences. These sequences are valuable sources to analyze not only the frequencies of certain events, but also the patterns with which these events happen. For example, from data consisting of web clicks one may discover that a large number of web browsers who visit *www.washingtonpost.com* in morning hours also visit *www.cnn.com*. Using such information one can group users as daily morning users, daily evening users, weekly users etc. This information might be useful for communicating to the users. Temporal patterns in the stock market, such as whether certain months, days of the week, time periods or holidays provide better returns than other time periods have received particularly a large amount of attention. Due to the presence of various types of applications in many fields, periodic pattern mining is an interesting area of study.

Mahanta et al. (2008) used set superimposition (Baruah, 1999) to find the membership value of each fuzzy interval. The concept of set *superimposition* is defined as follows. If set *A* is

superimposed over set B or set B is superimposed over set A then set superimposition operation can be expressed as A (S) $B = (A - B)$ $(+)$ $(A \cap B)^{(2)}$ $(+)$ $(B - A)$, where (S) denotes the set superimposition operation. Here, $(A \cap B)^{(2)}$ are the elements of $(A \cap B)$ represented twice and $(+)$ represents union of disjoint sets. Authors have also designed an algorithm for mining calendar-based periodic pattern. While applying this concept authors have assumed intervals with equal membership grade, and accordingly the concept of certainty factor has been proposed for each sub-interval. Certainty factor of an interval over different time periods expresses the likelihood of reporting the pattern in that particular interval. If two intervals overlap then the certainty factor is more for the overlapped region than the non-overlapped region. When two intervals are superimposed, authors have assumed 1/2 membership grade for each interval. After superimposition, the fuzzy membership value for the overlapped region becomes 1. The fuzzy membership value for non-overlapped region remains 1/2. But these two intervals may have different supports of the pattern. The certainty factor and support of a pattern in an interval are two different concepts. For better analysis of overlapped regions, these two concepts need to be introduced along with the overlapped region. Thus, in this chapter we propose an extended analysis of superimposed intervals. The main weak point of the aforementioned paper (Mahanta et al., 2008) is that the concept of set superimposition is not necessary in the proposed algorithm. Therefore, we have proposed an algorithm to identify full / partial calendar-based periodic patterns. We have also improved our algorithm by introducing a hash based data structure for storing relevant information associated with intervals. In addition, we have suggested some other improvements in the proposed algorithm. Before concluding this section, we take an example of

a time-stamped database that will be used for providing illustrative examples on various concepts.

Example 1. Consider the following database *D* of transactions. Each record contains items purchased as well as the date of the transaction.

Table 4.1 A sample time-stamped database

time-stamp	items	time-stamp	items	time-stamp	items
29/03/1990	*a, b, c*	07/04/1992	*a, c, e, g, h*	17/04/1993	*a, c, f*
06/04/1990	*a, c, e*	12/04/1992	*c, e*	06/04/1994	*a, b, c,d*
21/04/1990	*a, d*	14/04/1992	*c, e, f*	10/04/1994	*g, h*
25/04/1990	*a, c, d*	19/04/1992	*f, g*	13/04/1994	*a,* g
06/03/1991	*a, c*	04/03/1993	*a, c*	18/04/1994	*g, h, i*
12/03/1991	*a, c, e*	09/03/1993	*a, c, g*	20/04/1994	*a, c, e, f*
19/04/1991	*f, g*	01/04/1993	*c, h, i*		
03/03/1992	*a, c, d*	07/04/1993	*c, d*		

We have omitted the time of a transaction, since our data analysis is not associated with the time component of a transaction. We will refer to this database from time to time for the purpose of illustrating various concepts.

Rest of the chapter is organized as follows. We discuss related work in Section 4.2. In Section 4.3, we have discussed calendar-based periodic patterns and proposed an extended certainty factor of an interval. We have designed an algorithm for identifying calendar-based periodic patterns in Section 4.4. Experimental results are provided in Section 4.5. We conclude the chapter in Section 4.6.

4.2 Related work

A calendar time expression is composed of calendar units in a specific calendar and represents different time features, such as an absolute time interval and a periodic time over a specific time period. A calendar-based periodic pattern is associated with time hierarchy for calendar years. In this chapter we have dealt with calendar dates over the years.

Verma et al. (2005) have proposed an algorithm H-Mine, where a header table H is created separately for each interval. Each frequent item entry has three fields viz., an item-id, a support count and a hyper-link.

Lee et al. (2008) have proposed two data mining systems for discovering fuzzy temporal association rules and fuzzy periodic association rules. The mined patterns are expressed in fuzzy temporal and periodic association rules that satisfy the temporal requirements specified by the user. In the proposed algorithm the mined patterns are dependent on user inputs such as maximum gap between two intervals and minimum length of an interval.

Li et al. (2001) proposed two classes of temporal association rules, temporal association rules with respect to precise match and temporal association rules with respect to fuzzy match, to represent regular association rules along with their temporal patterns. A similar work was reported by Zimbrao et al. (2002). Authors incorporate multiple granularities of time intervals from which both cyclic and user-defined calendar patterns can be achieved. Ale and Rossi (2000) proposed an algorithm to discover temporal association rules. In this algorithm support of an item is calculated only during its lifespan.

Wenpo and Guanling (2006) have proposed a technique for mining partial multiple periodic patterns without redundant rules. Without mining every period, authors checked the necessary

period and used this information to do further mining. Instead of considering the whole database, the information needed for mining partial periodic patterns is transformed into a bit vector that can be stored in a main memory. This approach needs to scan the database at most two times. In the context of support definition Kempe et al. (2008) have proposed a new support definition that counts the number of pattern instances, handles multiple instances of a pattern within one interval sequence and allows time constraints on a pattern instance.

Lee et al. (2008) have proposed a new temporal data mining technique that can extract temporal interval relation rules from temporal interval data by using Allen's theory (Allen, 1983). Authors designed a preprocessing algorithm for generalization of temporal interval data. Also, authors have proposed an algorithm for discovering a temporal interval relation.

Ozden et al. (1998) proposed a method of finding patterns having periodic nature where the period has to be specified by the user. Han et al. (1999) proposed several algorithms for mining partial periodic patterns by exploring some interesting properties such as the apriori property and the max-subpattern hit set property by shared mining of multiple periods. Our approach is different from the methods described above. In our algorithm we consider the time-stamps as a hierarchical data structure and then extract periodic patterns for the different levels of hierarchy.

4.3 Calendar-based periodic patterns

In Sections 4.1 and 4.2, we have presented some important applications of calendar-based periodic patterns. A calendar-based periodic pattern is dependent on the schema of a calendar. There are various ways one could define the schema of a calendar. We assume that the schema of the calendar-based pattern is based on day, month and year. The calendar patterns based on a schema are not isolated, but related to each other. This schema is also useful to determine weekly-based pattern, since first seven days of any month correspond to the first week, days 8 to14 of any month correspond to the second week, and so on. Thus, one can have several types

of calendar-based periodic patterns viz., daily, weekly, monthly and yearly. Based on a schema, some examples of calendar patterns are given as follows: every day of January, 1999; every 16-th day of January in each year; second week of every month. Again, each of these periodic patterns could be of two types viz., partially periodic pattern and full periodic pattern. A problem related to periodicity could be of finding patterns occurring at regular time intervals. This concept emphasizes on two aspects viz., pattern and interval.

A calendar pattern refers to a market cycle that repeats periodically on a consistent basis. Seasonality could be a major force in the marketplace. While calendar patterns are based on a framework of multiple time granularities viz., day, month and year, but the periodic patterns are defined in term of a single granularity. These patterns are dependent on the lifespan of an item in a database. Lifespan of an item (x) is a pair (x, [t_1, t_2]), where t_1 and t_2 denote the time that the item x appears in the database for the first time and last time, respectively. The problem of periodic pattern mining can be categorized into two types. One is full periodic pattern mining, where every point in time granularity contributes to a cyclic behavior of the pattern. The other and more general one is called partial periodic pattern mining, which specifies the behavior of the pattern at some but not all points of time granularity in the database. Partial periodicity is a looser form of periodicity than full periodicity and it also occurs more commonly in the real world. A pattern is associated with a real number m ($0 < m < 1$), called match ratio (Li et al. 2003) that reveals a pattern holds with respect to fuzzy match satisfying at least $100m$% of the time intervals. Match ratio is an important measure which determines whether a calendar-based pattern could be full periodic or partial periodic. When the match ratio is equal to 1 then it is a full periodic pattern. In case of partial periodic pattern the match ratio lies between 0 and 1.

While finding yearly periodic patterns, Mahanta et al. (2008) have proposed match ratio in somewhat a different way. Authors have proposed match ratio as the number of intervals is divided by number of years in the lifespan of the pattern. It might be difficult to work with this definition, since a mining algorithm returns itemsets and their intervals. A mining algorithm might not be concerned with reporting the first and last appearances of an itemset. Therefore, we will follow the definition proposed by Li et al. (2003).

We have discussed the concept of certainty factor in Section 4.1. Also we have noticed that the analysis of overlapped region using certainty factor might not be sufficient. Therefore, we propose an extension to it.

4.3.1 Overlapped intervals

In this chapter we deal with overlapped intervals in different contexts. Let us consider that *A* and *B* are two overlapping intervals. By employing Allen's interval relations (Allen, 1983) one can have the following relationships between *A* and *B*: *A* before *B*, *A* meets *B*, *A* overlaps *B*, *A* is-finished-by *B*, *A* contains *B*, *A* is-started-by *B* and *A* equals *B*. Similarly there exists inverse relations from *B* to *A*. These relationships are pair wise different and are illustrated in Figure 4.1.

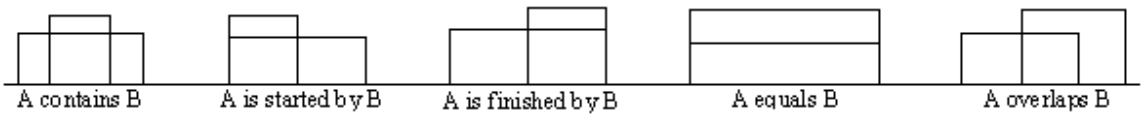

Figure 4.1 Different types of overlapped intervals

For finding calendar-based yearly patterns, each of the two intervals in Figure 4.1(a) corresponds to the same year. In the wider interval *A*, a certain pattern *X* is frequent, whereas another pattern *Y* is frequent in narrower interval *B*. We may be interested whether a higher-level pattern *XY* is frequent in the intersection of intervals *A* and *B*. For analyzing a calendar-based yearly pattern using the concept of certainty factor, we require multiple intervals where the pattern is frequent in different years. In this case each interval corresponds to a year. Thus, for a particular pattern there are multiple intervals. From the perspective the lifespan of the yearly pattern, these intervals might overlap. Thus, the concept of certainty factor is also associated with overlapped intervals.

4.3.2 Extending certainty factor

The concept of certainty factor is based on the concept of set superimposition. If we are interested in yearly patterns, during the analysis of superimposed intervals the year component is ignored. We explain here the concept of set superimposition using the following example.

Example 2. Consider the database of Example 1. Itemset {*a*, *c*} is present in 3 out of 4 transactions in the intervals [29/03/1990 - 25/04/1990]. Also, {*a*, *c*} is present in 2 out of 3 transactions in the intervals and [06/03/1991 - 19/04/1991]. Therefore, {*a*, *c*} is frequent in these intervals at minimum support level 0.66. These two intervals are being superimposed where each of these intervals has fuzzy set membership value 1/2. The overlapped area of these two intervals is [29/03 - 19/04]. Based on the concept of set superimposition, an itemset reported in a non-overlapped region has the fuzzy set membership value 1/2. But, an itemset reported in the overlapped interval [29/03 - 19/04] has fuzzy set membership value 1/2 + 1/2 = 1.

For the purpose of mining periodic patterns, Mahanta et al. (2008) have proposed certainty factor. It is based on a set of overlapped intervals corresponding to a pattern occurring on a periodic basis. For example, one might be interested in identifying yearly periodic patterns in a database. Authors have considered all the intervals having equal membership grade. For example, if n intervals are superimposed then every interval has $1/n$ equal membership grade and in an overlapped area the membership value will be added. The certainty of the pattern in the overlapped subinterval is more than the certainty in the other subintervals. Let $[t_1, t'_1]$ and $[t_2, t'_2]$ be two overlapped intervals where a pattern X gets reported with certainty value 1/2. When the two intervals are superimposed the certainty factors of X associated with the various subintervals are given as follows:

$$[t_1, t'_1]^{1/2}(S)\,[t_2, t'_2]^{1/2} = [t_1, t_2)^{1/2}\,[t_2, t'_1]^{1}\,(t'_1, t'_2]^{1/2} \quad \ldots(1)$$

The notion of certainty factor seems to be an important contribution made by the authors. It represents the certainty of reporting a pattern in an interval by considering a sequence of periods. For example, we might be interested in knowing the certainty of pattern $\{a, c\}$ in the month of April with respect to the database in Example 1. It is an important statistical evidence of a pattern in an interval over a sequence of years (periods). For example, one could say that the evidence of the pattern $\{a, c\}$ is certain in the month of April when the years viz., 1990, 1991, 1992 and 1993 are considered. But the concept of certainty factor does not convey the information regarding the frequency of a pattern in an overlapped region. In addition, it gives equal importance to all the intervals by considering them as equi-fuzzy intervals. From the perspective of the evidence of a pattern, such assumption might be realistic. But from the perspective of the depth of evidence,

such concept might not be sufficient. Thus, we propose an extension to the concept of certainty factor. In the proposed extension, we incorporate the information regarding support of a pattern in an interval. There are many ways one could keep the information regarding support. In Example 1, there are four overlapping intervals corresponding to the pattern $\{a, c\}$. There exists a region where all the intervals are overlapped. On the contrary, some regions may not be overlapped at all. Apart from the certainty factor of a region, one could also keep the support information of the pattern in that interval. In general, a region could be overlapped by all intervals. Let there are be n supports of a pattern corresponding to n intervals. Then the question comes to our mind, how to keep the support information of the pattern for n intervals. The answer to this question might not be agreeable to all. One might be interested in keeping the average support of the pattern along with the certainty factor for that interval. Some of us might be interested in keeping information regarding the minimum and maximum of n supports. In an extreme case, one might be interested in keeping all the n supports of the pattern corresponding to n intervals. Let us consider that we are interested in yearly pattern. Let the life-span of a pattern be forty years. Then one has to keep a maximum of forty supports corresponding to an overlapped region. It might not be realistic to maintain all the forty supports. Let *s-info*(X, $[t_1, t_2]$) be the support information of the pattern X for the interval $[t_1, t_2]$.

The *support* (Agrawal et al., 1993) of a pattern X represents a fraction of transactions containing X. A pattern X is *frequent* if its support is greater than equal to a user-defined threshold, *minsupp*. Let the pattern X be frequent in time intervals $[t_i, t'_i]$, $i = 1, 2, \ldots, n$. Each of these intervals is taken from a different period of time such that $\bigcap_{i=1}^{n} [t_i, t_i^{'}] \neq \varphi$. In Example 1,

patterns $\{a\}$, $\{c\}$ and $\{a, c\}$ get reported in the month of April in every year. By generalizing (1), the certainty factor of X in overlapped regions could be obtained as follows:

$$[t_1, t'_1]^{1/n} (S) [t_2, t'_2]^{1/n} (S) \dots (S) [t_n, t'_n]^{1/n} = [t^{(1)}, t^{(2)})^{1/n} [t^{(2)}, t^{(3)})^{2/n} [t^{(3)}, t^{(4)})^{3/n} \dots [t^{(r)}, t^{(r+1)})^{r/n} \dots \times$$

$$[t^{(n)}, t'^{(1)}]^1 (t'^{(1)}, t'^{(2)}]^{n-1/n} \dots (t'^{(n-2)}, t'^{(n-1)}]^{2/n} (t'^{(n-1)}, t'^{(n)}]^{1/n} \quad \dots(2)$$

where $\{t^{(i)}\}_{i=1}^{n}$ is the sequence obtained from $\{t_i\}_{i=1}^{n}$ by sorting in ascending order and $\{t'^{(i)}\}_{i=1}^{n}$ is obtained from $\{t'_i\}_{i=1}^{n}$ by sorting in ascending order. We propose an extended certainty factor of X in the above overlapped intervals as follows:

When X is reported in $[t^{(n)}, t'^{(1)}]$ then the certainty value is 1 with support information *s-info*(X, $[t^{(n)}, t'^{(1)}]$). But, the certainty value of X for the outside of $[t^{(1)}, t'^{(n)}]$ is 0 with support information 0. When X is reported in $[t^{(r-1)}, t^{(r)})$, then the certainty value is $(r - 1) / n$ with support information *s-info*(X, $[t^{(r-1)}, t^{(r)})$), for $r = 2, 3, \dots, n$. Otherwise, the certainty values of X for $(t'^{(r-1)}, t'^{(r)}]$ is $(n - r + 1) / n$ with support information *s-info*(X, $(t'^{(r-1)}, t'^{(r)}]$), for $r = 3, 4, \dots, n$.

Suppose we are interested in identifying yearly periodic patters. So each time interval is taken from a year. From the perspective of n years, the pattern X gets reported in every year in the interval $[t^{(n)}, t'^{(1)}]$. So, the certainty of X is 1 (highest) in this interval. But, X is not frequent pattern outside of $[t^{(1)}, t'^{(n)}]$. Therefore, from the perspective of all the years the certainty of X is 0 (lowest) outside of the interval. The certainty factor also provides the information regarding how many intervals are overlapped on a sub-interval. For example, if the certainty factor of a sub-interval is 2/5, for given five intervals, then two intervals are overlapped on the sub-interval. On the other hand *s-info* provides the information regarding degree of frequency of X in an interval. To illustrate the above concept we consider the following example.

Example 3. Although it is not based on the database given in Example 1, but it explains the proposed concept of extended certainty factor stated above. Let the years 1990, 1991, 1992 and 1993 be of our interest. We would like to check whether the pattern X is yearly periodic. Assume that the mining algorithm has reported X is frequent in the time intervals $[t_1, t'_1]$, $[t_2, t'_2]$, $[t_3, t'_3]$ and $[t_4, t'_4]$ for the years 1990, 1991, 1992 and 1993, respectively. Also, let the supports of X in $[t_1, t'_1]$, $[t_2, t'_2]$, $[t_3, t'_3]$ and $[t_4, t'_4]$ be 0.2, 0.15, 0.16 and 0.12, respectively. Based on the proposed extended concept, we wish to analyze the time interval $[t_1, t'_4]$ by overlapping these intervals corresponding to the four years. The overlapped intervals are depicted in Figure 4.2.

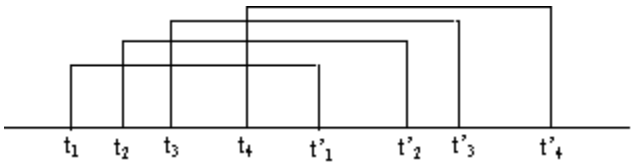

Figure 4.2 Overlapped intervals for finding yearly pattern X

While computing support information we use here the range measure for a set of values. One could use another support information depending on the requirement. An analysis of the overlapped intervals corresponding to X is presented in Table 4.2. Certainty of a sub-interval is based on the number of intervals overlapped on it. For example, $[t_1, t_2)$ has certainty 1/4, since there is only one interval out of four intervals.

Table 4.2 An analysis of the overlapped intervals for finding yearly pattern X

interval	certainty factor	*s-info*	interval	certainty factor	*s-info*
$[t_1, t_2)$	1/4	0.2 - 0.2	$(t'_1, t'_2]$	3/4	0.12 - 0.15
$[t_2, t_3)$	1/2	0.15 - 0.2	$(t'_2, t'_3]$	1/2	0.12 - 0.16
$[t_3, t_4)$	3/4	0.15 - 0.2	$(t'_3, t'_4]$	1/4	0.12 - 0.12
$[t_4, t'_1]$	1	0.12 - 0.2			

Here *s-info* corresponding to interval [t_3, t_4) represents the fact that the maximum and minimum supports of overlapped intervals are 0.2 and 0.15 respectively.

Certainty factor and support information are not the same. They represent two different aspects of a pattern in an interval. Certainty factor is normally associated with multiple time intervals. It expresses the likelihood of reporting a pattern in a sub-interval of the multiple overlapped intervals. But the concept of support is associated with a single time-interval. It is defined as the fraction of the transactions containing the pattern. Thus, for an effective analysis of a superimposed interval both certainty factor and support information are needed in association with an interval.

4.3.3 Extending certainty factor with respect to other intervals

In Figure 4.1 we have shown four intervals overlapped corresponding to four different years.

But in reality the scenario could be different. For four intervals, there may exist different combinations of overlapped intervals. But, whatever may be the case, the certainty factor of a sub-interval depends on the number of intervals overlapped in that sub-interval and *s-info* depends on the supports of the pattern in the intervals that are being overlapped on a sub-interval. Let us consider a sub-interval [*t, t*'], where *m* out of *n* intervals are overlapped on [*t, t*']. Based on certainty factor (Mahanta et al., 2008), we propose an extended certainty factor as follows:

When *X* is reported in [*t, t*'], then the certainty value is *m*/*n* with support information *s-info*(*X*, [*t, t*']), where *s-info*(*X*, [*t, t*']) is based on supports of *X* in the *m* intervals overlapped on [*t, t*']. We illustrate this issue with the help of Example 4. Before that, we present a few definitions related to overlapped intervals. Let *maxgap* be the user-defined maximum gap (time units) between

current time-stamp of a pattern and the time-stamp of the pattern when it was last seen. If the gap between current time-stamp of a pattern and the time-stamp of the pattern when it was last seen is greater than *maxgap* then a new interval is formed for the pattern with the current time-stamp as the start of the interval. Also, the previous interval of the pattern was ended when it was last seen. Let *mininterval* be the minimum period length of a time interval. Each interval should be of sufficient length, otherwise a pattern appearing once in a transaction is also become frequent in an interval. If two intervals are overlapped and the length of the overlapped region exceeds *mininterval* then the overlapped region could be interesting, otherwise it is discarded.

Example 4. We refer to the database of Example 1. Let the value of *maxgap* be 40 days. Then pattern {*a*, *c*} gets reported in the following intervals : [29/03/1990 - 25/04/1990], [06/03/1991 - 12/03/1991], [03/03/1992 - 07/04/1992], [04/03/1993 - 17/04/1993], and [06/04/1994 – 20/04/1994]. Let the value of *mininterval* be 10 days. The interval [06/03/1991 - 12/03/1991] does not satisfy the criterion of *mininterval*. Also let the value of *minsupp* be 0.5. Then {*a*, *c*} is not locally frequent in the interval [06/04/1994 - 20/04/1994]. We shall analyse the pattern {*a*, *c*} in the following intervals: [29/03/1990 - 25/04/1990], [03/03/1992 - 07/04/1992], and [04/03/1993 - 17/04/1993]. After superimposition, we require to analyse the interval [03/03 - 25/04]. We present superimposed intervals in Figure 4.3.

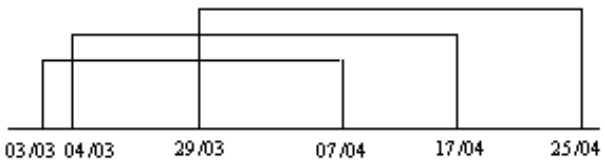

Figure 4.3 Overlapped intervals for finding yearly pattern {*a*, *c*}

We present an analysis of the time interval [03/03 - 25/04] based on the concept of extended certainty factor. Extended certainty factor of a pattern in an interval provides information of both certainty factor and *s-info* for the pattern. In Table 4.3 we present an analysis of intervals for finding yearly pattern {*a*, *c*}.

Table 4.3 An analysis of the time interval [03/03 - 25/04] for finding yearly pattern {*a*, *c*}

interval	certainty	*s-info*	interval	certainty	*s-info*
[03/03 - 04/03)	1/5	1.0 - 1.0	(07/04 - 17/04]	2/5	0.6 - 0.75
[04/03 - 29/03)	2/5	0.6 - 0.75	(17/04 - 25/04]	1/5	0.75 - 0.75
[29/03 - 07/04]	3/5	0.6 - 1.0			

In the above we have presented an analysis of the time interval [03/03 - 25/04]. The subintervals [03/03 - 04/03) and (17/04 - 25/04] are also shown, but they do not satisfy *mininterval* criterion. In the experimental results we have not presented such subintervals.

4.4 Mining calendar-based periodic patterns

Itemsets in transactions could be considered as a basic type of pattern in a database. Many interesting patterns like association rules (Agrawal et al., 1993), negative association rules (Wu et al., 2004), Boolean expressions induced by itemset (Adhikari & Rao, 2007) and conditional patterns (Adhikari & Rao, 2008) are based on itemset patterns. Some itemsets may be frequent in certain time intervals but may not be frequent throughout the lifespan of the itemsets. In other words, some itemsets may appear in the transactions for a certain time period and then disappear for a long period and then reappear. In view of making a data analysis involving various

itemsets, it might be required to extract the itemsets together with the time-slots in which they are frequent.

4.4.1 An overview of calendar-based periodic pattern

Calendars are typically used to describe events or time related properties over the same span of time using different granularities. A granule is a set of time instants perceived as a non-decomposable temporal entity when used to describe a phenomenon or, in general, when used to time-stamp a set of data (Bettini et al., 2000). For example, the Gregorian calendar comprises the granularities day, month and year. A granule can be composed of a single instant, a set of contiguous instants (time interval) or a set of noncontiguous instants.

4.4.2 Improving mining calendar-based periodic patterns

The goal of this chapter is to study the existing algorithm, and to propose an effective algorithm by improving the limitations of existing algorithm for mining calendar-based periodic patterns. We have discussed earlier that the concept of certainty factor of an interval does not provide good analysis of overlapped intervals. Therefore, the concept of extended certainty factor has been proposed. In view of designing an effective algorithm, we also need to understand the existing algorithm. While studying the existing algorithm (Mahanta et al., 2005) we have found that some variables contradict their definitions. The Algorithm 4.1 (Mahanta et al., 2005) finds all the locally frequent itemsets of size one. Authors defined two variables *ptcount* and *ctcount* as follows. The variable *ptcount* is used to count the number of transactions in an interval in which the current item belongs. On the other hand, the variable *ctcount* is used to count the number of transactions in that interval. Therefore, the assignment *ptcount*[*k*] = *ctcount*[*k*] in Algorithm 4.1, is not correct. Also, the variable *icount* is defined as the number of items present in the whole

dataset. Therefore, the initialization, *icount* = 1, placed just before starting a new interval seems to be not appropriate. Moreover, the validity of the experiment is low, since the experimental results are based on only one dataset. We propose a number of improvements mentioned as follows: (i) The proposed algorithm makes corrections on the existing algorithm based on the points mentioned above. (ii) It makes effective data analysis by incorporating extended certainty factor. (iii) We propose a hash-based data structure to improve the space efficiency of our algorithm. (iv) Also, we have improved the average time complexity of the algorithm. (v) We make a comparative analysis with the existing algorithm. (vi) In addition, we have improved the validity of the experimental results by conducting experiments on more datasets.

4.4.3 Data structure

We discuss here the data structure used in the proposed algorithms for mining itemsets along with the time intervals in which they are frequent. A hash-based data structure is a natural way of storing different itemsets and their associated information. Since there are number of yearly databases, all the frequent itemsets in a particular year should be linked corresponding to that year. We describe the data structure using the following example.

Example 5. Consider the database *D* of Example 1. Transactions are made in the years 1990, 1991, 1992, 1993, and 1994, where the items *a, b, c, d, e, f, g, h,* and *i* appear in *D*. We propose Algorithm 1 to mine locally frequent itemsets of size one along with their intervals. The algorithm produces output as shown at level 1 of Figure 4.4. We assume here *maxgap, mininterval* and *minsupp* as 40 days, 5 days and 0.5, respectively. We are interested in identifying yearly periodic patterns. At the level 0 we have shown all the years that appeared in the transaction. The pointer corresponding to the year 1990 keeps all the locally frequent

itemsets of size one, their supports and their intervals. All the five years are stored in an index table at level 0. After level 0, we keep an array of pointers of each year. The first pointer corresponding to year 1990 points to a table containing interesting itemsets of size one for the year 1990, their intervals and their local supports. The second pointer corresponding to year 1990, points to a table containing interesting itemsets of size two for the year 1990, their intervals and their local supports, and so on. Here itemsets of size three corresponding to a year do not get reported. Different itemsets, their intervals, and supports are shown in Figure 4.4.

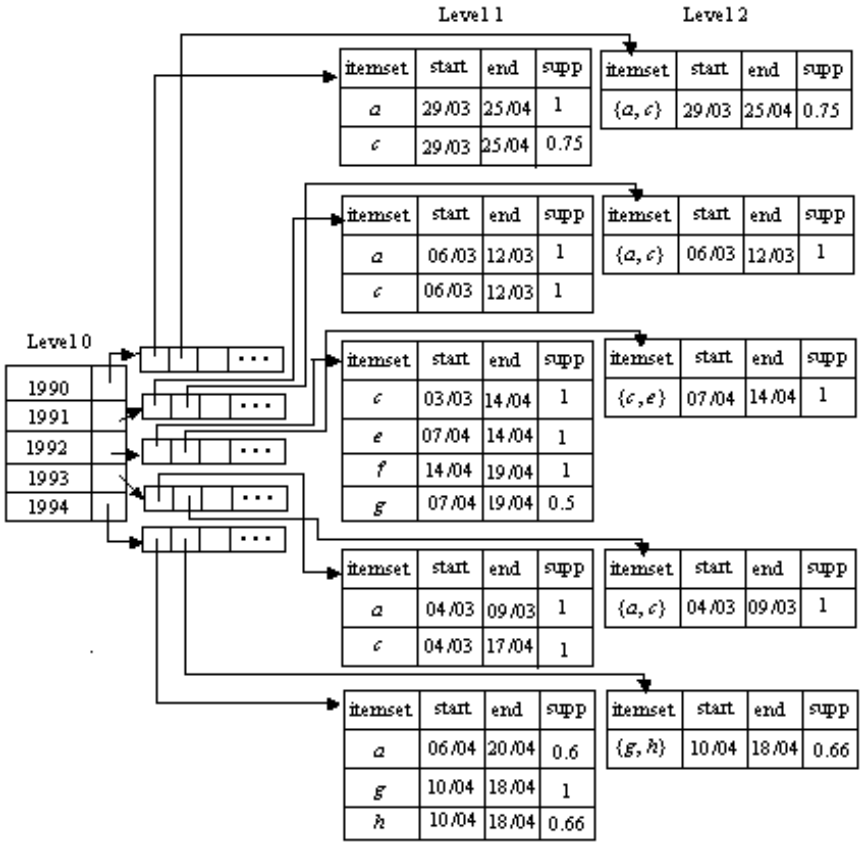

Figure 4.4 Data structure used in the proposed algorithms

4.4.4 A modified algorithm

As mentioned in Section 4.1, we have proposed a number of improvements to the existing algorithm, Algorithm 4.1 (Mahanta et al., 2005), for finding locally frequent itemsets of size one. We calculate the support of each item in an interval and store it whenever the item is frequent in that interval. Intervals that satisfy the user-defined constraint *mininterval* are retained corresponding to the itemsets of size one that satisfy the user-defined constraint *minsupp*. The modification made seems to be significant from the overall view point of Apriori algorithm. We have used a hash-based data structure to improve efficiency of storing and accessing locally frequent itemsets of size one. We explain here all the variables and their functions in the following paragraph.

Let *item* be an array of items in *D*. Also let the total number of items be *n*. We use index *level_0* to keep track of different years. It is a two-dimensional array containing 2 columns. First column of *level_0* contains the different years in increasing order. A two-dimensional array *itemset_addr* is used to store the addresses of tables containing itemsets. *itemset_addr*[*row*][*j*] contains the address of the table containing locally frequent itemsets of size *j* for the current year *row*. The second column of *level_0* stores addresses of arrays pointing to these tables. Tables at *level_p* store the frequent itemsets of size *p*, $p = 1, 2, 3, \ldots$. Variables *row* and *row_p* are used to index arrays *itemset_addr* and *level_p* respectively, $p = 0, 1, 2, \ldots$. We consider a transaction as a record containing transaction date (*date*) and items purchased. Function *year*( ) is used to extract year from a given date. *firstseen*[*k*] and *lastseen*[*k*] specify the date when the *k*-th item is seen for the first time and last time in an interval, respectively. Each item in the database is associated with the arrays *itemIntervalFreq* and *nTransInterval*. Cells *itemIntervalFreq*[*k*] and

nTransInterval[*k*] are used to keep the number of transactions containing item *k* and total number of transactions in a time interval, respectively. Variable *nItemsTrans* is used to keep track of the number of items in the current transaction. The goal of the proposed algorithm is to find all the locally frequent itemsets of size one, their intervals and supports. The algorithm is presented as follows.

Algorithm 1. Mine locally frequent items and their intervals

procedure *MiningFrequentItems_One* (*D, maxgap, mininterval, minsupp*)

Inputs: *D, maxgap, mininterval, minsupp*

D: database to be mined

minsupp: as defined in Section 4.3.2

maxgap, mininterval: as defined in Section 4.3.3

Outputs:

Locally frequent items, their intervals and supports as mentioned in Figure 4.4

01: **let** *nItemsTrans* = 0; *row* = 1; *row_0* =1; *row_1* = 1;

02: **for** *k* = 1 to *n* **do**

03: *lastseen*[*k*] = 0; *itemIntervalFreq*[*k*] = 0; *nTransInterval*[*k*] = 0;

04: **end for**

05: read a transaction $t \in D$;

06: *level_0*[*row_0*][1] = *year*(*t.date*);

07: *level_0*[*row_0*][2] = *itemset_addr*[*row*][1];

08: **while** not end of transaction in *D* **do**

09: *transLength* = |*t*|;

```
10:   if (level_0[row_0][1] ≠ year(t.date)) then
11:     for k = 1 to n do
12:       if (|lastseen[k] – firstseen[k]| ≥ mininterval) and
          (itemIntervalFreq[k] / nTransInterval[k] ≥ minsupp) then
13:         level_1[row_1][1] = item[k]; level_1[row_1][2] = firstseen[k];
14:         level_1[row_1][3] = lastseen[k];
15:         level_1[row_1][4] = itemIntervalFreq[k] / nTransInterval[k];
16:         increase row_1 by 1;
17:       end if {12}
18:     end for {11}
19:     row_1 = 1;
20:     increase row_0 by 1; increase row by 1;
21:     level_0[row_0][1] = year(t.date); level_0[row_0][2] = itemset_addr[row][1];
22:     for k = 1 to n do
23:       lastseen[k] = 0; itemIntervalFreq[k] = 0; nTransInterval[k] = 0;
24:     end for
25:   end if {10}
26:   for k = 1 to n do
27:     if (item[k] ∈ t) then
28:       increase nItemsTrans by 1;
29:       if (lastseen[k] = 0) then
30:         initialize both lastseen[k] and  firstseen[k] by t.date;
```

initialize both *itemIntervalFreq*[*k*] and *nTransInterval*[*k*] by 1;

else if (| *t.date* – *lastseen*[*k*] | ≤ *maxgap*) **then**

lastseen[*k*] = *t.date*;

increase *itemIntervalFreq*[*k*] by 1; increase *nTransInterval*[*k*] by 1;

end if

else if (| *lastseen*[*k*] – *firstseen*[*k*] | ≥ *mininterval*) **and**

(*itemIntervalFreq*[*k*] / *nTransInterval*[*k*] ≥ *minsupp*) **then**

level_1[*row_1*][1] = *item*[*k*]; *level_1*[*row_1*][2] = *firstseen*[*k*];

level_1[*row_1*][3] = *lastseen*[*k*];

level_1[*row_1*][4] = *itemIntervalFreq*[*k*] / *nTransInterval*[*k*];

increase *row_1* by 1;

initialize both *lastseen*[*k*] and *firstseen*[*k*] by *t.date*;

initialize both *itemIntervalFreq*[*k*] and *nTransInterval*[*k*] by 0;

end if {35}

end if {29}

else increase *nTransInterval*[*k*] by 1;

end if {27}

if (*nItemsTrans* = *transLength*) **then exit** from *for-loop*; **end if**

end for {26}

read a transaction *t* ∈ *D*;

end while {08}

for *k* = 1 to *n* **do**

51: **if** (|*lastseen*[*k*] – *firstseen*[*k*]| ≥ *mininterval*) **and**

(*itemIntervalFreq*[*k*] / *nTransInterval*[*k*] ≥ *minsupp*) **then**

52: *level_1*[*row_1*][1] = *item*[*k*]; *level_1*[*row_1*][2] = *firstseen*[*k*];

53: *level_1*[*row_1*][3] = *lastseen*[*k*];

54: *level_1*[*row_1*][4] = *itemIntervalFreq*[*k*] / *nTransInterval*[*k*];

55: increase *row_1* by 1;

56: **end if** {51}

57: **end for** {50}

58: sort arrays *level_1* on non-increasing order on primary key item and secondary key start date;

end procedure

At line 5 we read the first transaction of database. Afterwards the first row of the index *level_0* is initialized with the first year obtained from the transaction. The pointer field of the first row of *level_0* is initialized by the address of the first row of the table *itemset_addr*. Lines 8-46 are repeated until all the transactions are read. At line 10 we check whether the current transaction belongs to a different year. If it happens so then we close the last interval of different items using lines 11-18. We retain those intervals that satisfy criteria of *mininterval* and *minsupp*. Lines 19-24 assign the necessary initializations for a different year. Lines 26-45 are repeated for each item in the current transaction. Line 29 checks whether the item is first time seen in the transaction and the necessary assignment is done in line 30. Lines 31-34 determine whether the current transaction-date is coming under the current interval by comparing the difference between *t.date* and *lastseen* with *maxgap*. Lines 35-42 construct an interval and compute the local support.

Line 46 avoids the unnecessary repetition by comparing the transaction length. Line numbers 50-57 close all the last intervals for last year. Line 58 sorts arrays *level_1* on non-increasing order on primary key item and secondary key start date.

The time complexity of the algorithm has been reduced significantly by computing the length of current transaction (at line number 9) and putting a check at line number 27. Consider a database containing 10,000 items. Let the current transaction be of length 20 and these 20 items are within the first 100 items. Then the *for-loop* at line number 26 need not have to continue for the remaining 9,900 items, but the worst-case complexity of the algorithm remains the same as before.

We shall now present below an algorithm that makes use of locally frequent itemsets obtained by Algorithm 1 and apriori property (Agrawal & Srikant, 1994). We use array *level_1* to generate the candidate sets at second level. Then array *level_2* is used to generate candidate sets at the third level, and so on. We apply pruning using conditions at line 6 to eliminate some itemsets at the next level. This pruning step ensures that the size of the itemsets at the current level is one more than the size of an itemset at the previous level. Also we apply pruning using user-defined thresholds at line 13.

Algorithm 2. Mine locally frequent itemsets at higher level and their intervals

procedure *MiningHigherLevelItemsets* (*D*, *S*)

Inputs: *D, S*

D: database to be mined

S: partially constructed data structure containing locally frequent itemsets of size one

Outputs: locally frequent itemsets at higher levels, their intervals and supports as mentioned in Figure 4.4

01: **let** L_1 = set of elements at *level_1*of *S*; **let** $k = 2$;

02: **while** $L_{k-1} \neq \phi$ **do**

03: $C_k = \phi$;

04: **for** each itemset $l_1 \in L_{k-1}$ **do**

05: **for** each itemset $l_2 \in L_{k-1}$ **do**

06: **if** $((l_1[1] = l_2[1]) \wedge \ldots \wedge (l_1[k\text{-}2] = l_2[k\text{-}2]) \wedge (l_1[k\text{-}1] < l_2[k\text{-}1]))$ **then**

07: $c = l_1 \bowtie l_2$; $C_k = C_k \cup c$;

08: **end if** {06}

09: **end for** {05}

10: **end for** {04}

11: **for** each element $c \in C_k$ **do**

12: construct intervals for *c* as mentioned in Algorithm 1;

13: **if** the intervals corresponding to *c* satisfy *maxgap*, *mininterval* and *minsupp* **then**

14: add *c* and the intervals to *level_k* of *S*;

15: **end if** {13}

16: **end for** {11}

17: increase *k* by 1;

18: **let** L_k = set of elements at *level_k* of *S*;

19: **end while** {02}

end procedure

Using Algorithms 1 and 2, one could construct the data structure *S* presented in Figure 4.4 completely. Now we shall use *S* to determine whether an itemset pattern is fully / partially periodic. For this purpose we present Algorithm 3 in the following. It reports all the fully periodic itemsets as well as a subset of partially periodic itemsets. The variable *noi* represents the number of intervals overlapped. We are interested in the partially periodic itemsets if the certainty factor is greater than or equal to a user-defined threshold value (μ). In case of fully periodic itemsets μ is 1. Collections *F* and *P* store all the fully and partially periodic itemsets respectively.

Algorithm 3. Determining periodicity of itemsets

procedure *Periodicity* (*S*, μ)

Inputs:

S: data structure containing locally frequent itemsets, their intervals, and supports

μ: user-defined threshold of periodicity

Outputs:

Interesting periodic itemsets

01: **let** $k = 1$; **let** L_1 = elements at *level_1* of *S*; **let** $F = \phi$; $P = \phi$;

02: **while** $L_k \neq \phi$ **do**

03: **for** each element *l* of L_k **do**

04: **let** $i_1, i_2, \ldots, i_r$ be the intervals corresponding to *l*;

05: *startYear* = year of first interval of *l*; *endYear* = year of last interval of *l*;

06: **let** *olr* be an overlapped region among intersecting intervals;

07: **while** there exists a *olr* **do**

08: **if** (|*olr*| ≥ *mininterval*) **then**

09: **if** (*r* = *endYear* - *startYear* + 1) **then**

10: add *l* and the details of the overlapped interval to *F*;

11: **else if** (number of intersecting intervals / *r* ≥ μ) **then**

12: add *l* and the details of the overlapped interval to *P*;

13: **end if** {11}

14: **end if** {09}

15: **end if** {08}

16: **let** *olr* be another overlapped region among intersecting maximal intervals;

17: **end while** {07}

18: **end for** {03}

19: increase *k* by 1;

20: **let** L_k = set of elements at *level_k* of *S*;

21: **end while** {02}

22: **for** each element $l \in F \cup P$ **do**

23: display subintervals of *l*, their certainty factors and support information;

24: **end for** {22}

end procedure

We process the itemsets level-wise. Periodicities of itemsets at level one are checked using lines 2-21. At line 19 we move on to the next level. For each itemset at a given level may have more than one overlapped region where an overlapped region is generated by a set of overlapping

intervals. Therefore, lines 7-17 are repeated for overlapped region for a given itemset. In case of fully periodic pattern an overlapping interval is generated from every year. So the periodicity of a fully periodic pattern is 1 (highest). But an itemset may not have an interesting interval for a particular year. Thus, an overlapped region corresponding to a partially periodic pattern contains lesser number of intersecting intervals. At line 11 we check the periodicity of such patterns and consider only those patterns whose periodicities are greater than equal to μ. Interesting periodic itemsets are displayed in lines 22-24.

4.5 Experimental studies

We have carried out several experiments for mining calendar-based periodic patterns in different databases. All the experiments are performed on a 2.4 GHz, core i3 processor with 4 GB of memory, running Windows 7 HB, using Visual C++ (version 6.0) software. The data was stored on a 360 GB SATTA drive with 7000 rpm. We present experimental results using *retail* (Frequent itemset mining dataset repository), *BMS-WebView-1* (Frequent itemset mining dataset repository), and *T10I4D100K* (Frequent itemset mining dataset repository) databases. Since the records of these databases consist of only items purchased in a transaction, we have attached time-stamps randomly as calendar date for the transactions. The characteristics of the databases are given in Table 4.4.

Table 4.4 Database characteristics

D	*NT*	*ALT*	*AFI*	*NI*
retail	88,162	11.31	60.54	16,470
BMS-WebView-1	1,49,639	2.00	44.57	21,614
T10I4D100K	1,00,000	11.10	1276.12	870

Each of the databases *retail*, *BMS-WebView-1* and *T10I4D100K* has been divided into 30 sub-databases, called yearly databases, for the purpose of conducting experiments. The characteristics of these databases are given in Table 5. Let *D, NT, ALT, AFI,* and *NI* be the given database, the number of transactions, average length of a transaction, average frequency of an item, and the number of items, respectively. In Table 4.5 we have shown how the transactions have been time-stamped. The yearly databases obtained from *retail*, *BMS-WebView-1* and *T10I4D100K* are named as R_i, B_i and T_i respectively, $i = 1, \ldots, 30$. For simplicity, we have kept the number of transactions in each of the yearly databases fixed, except for the last database. We assume that the first and the last transactions occur on 01/01/1961 and 31/12/1990 respectively, and also assume that each year contains 365 days. In our experimental studies we report yearly periodic patterns and their periodicities in the above databases. We also compute certainty factor and match ratio of a pattern with respect to overlapped intervals. The density of *retail*, *BMS-WebView-1*, and *T10I4D100K* are 0.0007, 0.0003, and 0.013 respectively. All three databases are sparse.

In addition to partial periodic patterns, we mine full periodic patterns in the above databases. Itemset patterns of size one and two of *retail* is shown in Tables 4.6 and 4.7 respectively. In *retail* the itemsets {39} and {48} occur in all the thirty years and they are periodic throughout the year. Therefore, these itemsets are full periodic in the interval [1/1-31/12]. Itemset {41} is partially periodic, since the match ratio is less than 1. Initially it becomes frequent for thirteen years and then it does not get reported, and again it becomes frequent for the last six years. The subintervals that do not satisfy the *mininterval* criterion are not shown.

We have noticed some peculiarity in the mined patterns. For example, many patterns such as {0} and {1} are frequent throughout a year.

Table 4.5 Characteristics of yearly databases

D	*NT*	starting date, ending date	average number of transactions per day
R_1	2920	01/01/1961, 31/12/1961	8
…	…	…	…
R_{29}	2920	01/01/1989, 31/12/1989	8
R_{30}	3482	01/01/1990, 31/12/1990	9.54
B_1	5110	01/01/1961, 31/12/1961	14
…	…	…	…
B_{29}	5110	01/01/1989, 31/12/1989	14
B_{30}	1449	01/01/1990, 31/12/1990	3.97
T_1	3285	01/01/1961, 31/12/1961	9
…	…	…	…
T_{29}	3285	01/01/1989, 31/12/1989	9
T_{30}	4735	01/01/1990, 31/12/1990	12.97

Although, it is peculiar but it remains also an artificial phenomenon, since the time-stamps are enforced. There are many itemsets such as {16217} are frequent in many years with non-overlapping intervals. There are some items such as {647} and {769} are frequent twice in a year. In Figure 6, we present itemsets of size one that are also part of interesting itemsets of size two. While computing the certainty factor of an itemset we have used lifespan of the itemset. For example, itemset {0} gets reported from two years and it becomes frequent in both the years. Therefore its certainty factor is 2/2 = 1.

Table 4.6 Selected yearly periodic itemsets of size one (for *retail*)

retail (*minsupp* = 0.25, *mininterval* = 8, *maxgap* = 10)				
itemset	intervals	certainty	*s-info*	match ratio
{0}	[1/1-31/12]	2/2	0.35-0.66	1.0
{1}	[3/1-31/12]	2/3	0.57-0.66	0.67
{39}	[1/1-31/12]	30/30	0.52-0.63	1.0
{41}	[1/1-22/12]	13/30	0.26-0.32	0.43
{41}	[2/12-30/12]	6/30	0.27-0.32	0.20
{48}	[1/1-31/12]	30/30	0.43-0.53	1.0
{16217}	[1/1- 30/5]	1/1	0.87-0.87	1.0
{16217}	[7/9- 31/12]	1/1	0.97-0.97	1.0

Table 4.7 Yearly periodic itemsets of size two (for *retail*)

retail (*minsupp* = 0.25, *mininterval* = 8, *maxgap* = 10)				
itemset	intervals	certainty	*s-info*	match ratio
{0, 1}	[15/10-31/12]	1/1	0.46	1.0
{39, 41}	[1/1-30/12]	1/1	0.25	1.0
{39, 48}	[1/1-30/12]	30/30	0.28-0.38	1.0
{39, 16217}	[1/1- 30/5]	1/1	0.34	1.0
{48, 16217}	[7/9- 31/12]	1/1	0.27	1.0

Interesting itemset patterns of size one and two in *BMS-WebView-1* are shown in Tables 4.8 and 4.9 respectively. Here full periodic patterns are not reported since all the itemsets in *BMS-WebView-1* have match ratio less than 1. Therefore, these patterns are partial periodic. Itemset {12355} becomes frequent in three years but it has lifespan for seven years. In this database the items are sparse. Therefore, one requires choosing smaller *minsupp*. From Table 4.9 one could observe that itemset {33449, 33469} shows periodicity by appearing two times in six years and remaining interesting itemsets are reported for a year only.

Table 4.8 Yearly periodic itemsets of size one (for *BMS-WebView-1*)

BMS-WebView-1(*minsupp*=0.06, *mininterval* = 7, *maxgap* = 10)				
itemset	intervals	certainty	*s-info*	match ratio
{10311}	[29/1-6/10]	2/6	0.063 - 0.86	0.33
{12355}	[21/12-28/12]	3/7	0.060 - 0.061	0.43
{12559}	[22/4-11/5]	1/2	0.064 - 0.066	0.5
{33449}	[3/1-26/12]	5/7	0.063 - 0.08	0.71
{33469}	[3/1-31/3]	5/7	0.067 - 0.08	0.71

Table 4.9 Yearly periodic itemsets of size two (for *BMS-WebView-1*)

BMS-WebView-1(*minsupp*=0.06, *mininterval* = 7, *maxgap* = 10)				
itemset	intervals	certainty	*s-info*	match ratio
{10311, 12559}	[30/4-9/4]	1/1	0.06-0.06	1.0
{10311, 33449}	[3/3-11/4]	1/1	0.065	1.0
{33449, 33469}	[15/2-25/3]	2/6	0.061-0.064	0.33

In Table 4.10 we present yearly periodic itemsets of size one for *T10I4D100K* dataset. In this dataset patterns with full periodicity are not available, since the intervals corresponding to an item are not overlapped. We have presented examples of such items in the following table. From interval column one could observe that the itemsets are frequent for the short intervals, but do not appear at the same time for all the years. For example, itemset {966} appears in three intervals for 1961, but it does not show any periodicity since the intervals are not overlapped. It is interesting to note that the itemset {966} appears at the beginning, both in first and second months, of the year, then at the middle of the year i.e., for the third and fourth months, and finally at the end of the year (eleventh and twelfth month). This is also true for itemset {998}. Interesting itemset patterns of size two are not reported from this database.

An itemset that satisfies *minsupp*, *mininterval* criteria are reported. Also, a locally frequent itemset in two intervals for a particular year is also reported from the intervals, provided the

intervals satisfy *maxgap* criterion. The number of interesting intervals could increase by lowering the thresholds. In the following paragraphs we have presented a study on this aspect.

Table 4.10 Selected yearly periodic itemsets of size one (for *T10I4D100K*)

T10I4D100K (*minsupp*=0.13, *mininterval* = 7, *maxgap* = 10)					
itemset	interval	*s-info*	itemset	interval	*s-info*
{966}	[28/1/1961 - 19/2/1961]	0.16	{998}	[13/11/1964 - 22/11/1964	0.17
{966}	[16/3/1961 - 23/3/1961]	0.17	{998}	[15/12/1964 - 27/12/1964]	0.16
{966}	[14/12/1961 - 25/12/1961]	0.16	{998}	[2/9/1965 - 13/9/1965]	0.14
{966}	[22/3/1964 - 13/4/1964]	0.15	{998}	[27/11/1966 - 8/12/1966]	0.14
{966}	[1/11/1975 - 8/11/1975]	0.16	{998}	[27/11/1973 - 11/12/1973]	0.13
{966}	[12/4/1981 - 19/4/1981]	0.17	{998}	[14/12/1983 - 23/12/1983]	0.17
{966}	[2/12/1988 - 12/12/1988	0.15	{998}	[15/12/1984 - 23/12/1984]	0.16

4.5.1 Selection of *mininterval* and *maxgap*

The usage of constraints is application specific. Depending on the patterns that the user is targeting and also on the nature of the dataset, an optimal setting can be defined for the temporal constraints. This setting will result in the elimination of undesired patterns and at the same time provide further pruning.

The selection of *mininterval* and *maxgap* might be crucial since the process of data mining would depend on factors like seasonality, type of application and the data source. Some items are used for a particular season; while others are purchased throughout the year. When the items are

purchased throughout the year, the choices of *mininterval* and *maxgap* do not have much significance in mining yearly patterns. This observation seems to be valid for the items in *retail* and *BMS-WebView-1*. But the items in *T10I4D100K* are frequent in smaller intervals and therefore, *mininterval* and *maxgap* might have an impact on data mining. On the other hand, the requirement of an organization might determine an important parameter for mining calendar-based patterns. The distribution of items in databases also matters in selecting the right values of *mininterval* and *maxgap*. For a sparse database *maxgap* could be longer, and it could be even longer than *mininterval* provided *minsupp* remains small.

4.5.1.1 *Mininterval*

In the following experiments we would like to analyse the effect of *mininterval* for given *maxgap* and *minsupp*. We observe in Figures 4.6, 4.7 and 4.8, the number of intervals decreases as *mininterval* increases. An itemset might be frequent in many intervals. The number of itemsets frequent in an interval decreases as the length of *mininterval* increases. Although the above observation is true in general, but the type of the graphs might differ from one database to another. In *retail* many itemsets are locally frequent for longer periods of time. In Figure 4.6 we observe that there exist nearly 110 intervals for *mininterval* of 29 days. Whereas in *BMS-WebView-1* and *T10I4D100K*, the itemsets are frequent for shorter duration. As a result, the number of intervals reduces significantly when *mininterval* remains small. Thus the choice of *mininterval* is an important issue. In case of *retail* one could observe that initial decrement is slow with the increase of mininterval (upto 21), later there is a rapid fall in number of intervals. But the opposite thingh is observed in Figure 4.9, where intial decrement is sharp. Therefore,

here the items are locally frequent for longer period and the gaps between two intervals are shorter.

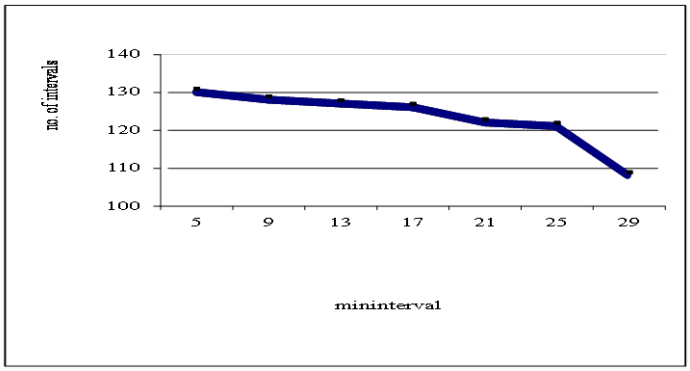

Figure 4.6 *retail* (*minsupp* = 0.25, *maxga*p = 7)

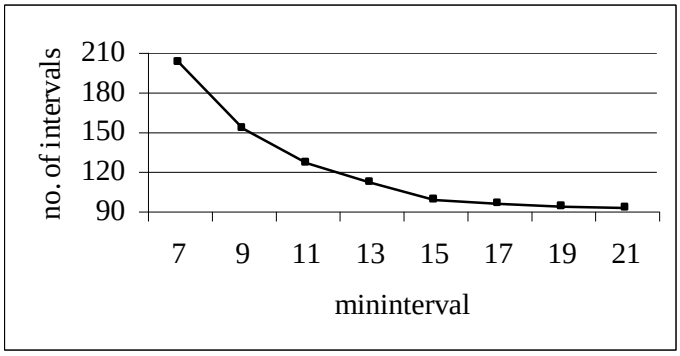

Figure 4.7 *BMS-WebView-1* (*minsupp* = 0.06, *maxga*p = 7)

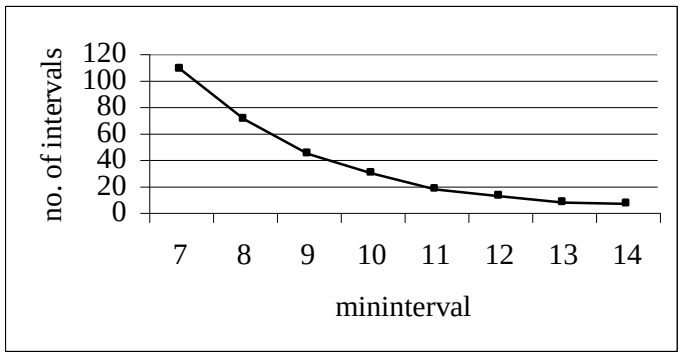

Figure 4.8 *T10I4D100K* (*minsupp* = 0.13, *maxga*p = 7)

4.5.1.2 *Maxgap*

In view of analyzing *maxgap* parameter, we now present graphs in Figures 4.9, 4.10, and 4.11 for the number of intervals versus *maxgap* at given *minsupp* and *mininterval*. The graphs show that the number of intervals decreases as *maxgap* increases. In *retail* the number of intervals decreases rapidly when *maxgap* varies from 5 to 10. Afterwards the change is not so significant. In *BMS-WebView-1* the decrement takes place almost at a uniform rate. Unlike *retail* and *BMS-WebView-1*, the number of intervals decreases faster at the smaller values of *maxgap* in *T10I4D100K* dataset.

4.5.2 Selection of *Minsupp*

The number of intervals and support of a database are inversely related at given *maxgap* and *mininterval*.

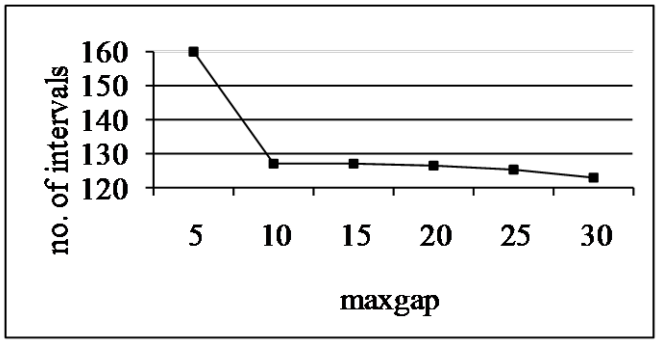

Figure 4.9 *retail* (*minsupp* = 0.25, *mininterval* = 10)

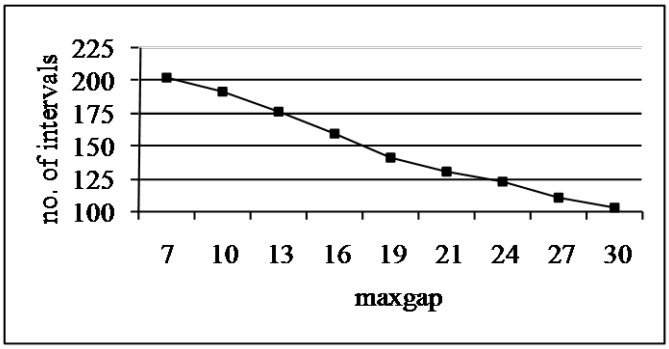

Figure 4.10 *BMS-WebView-1*(*minsupp* = 0.06, *mininterval* = 7)

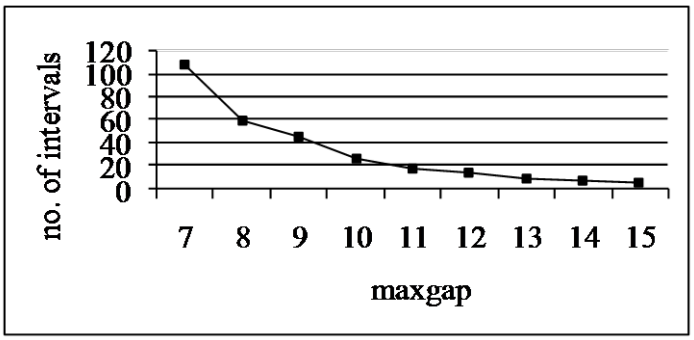

Figure 4.11 *T10I4D100K* (*minsupp* = 0.13, *mininterval* = 7)

We observe this phenomenon in Figures 4.12, 4.13 and 4.14. When the value of *minsupp* is smaller the number of intervals reported is quite large. Initially the number of intervals reported significantly with small decrement of *minsupp*. Later the decrement of number of intervals is not so significant.

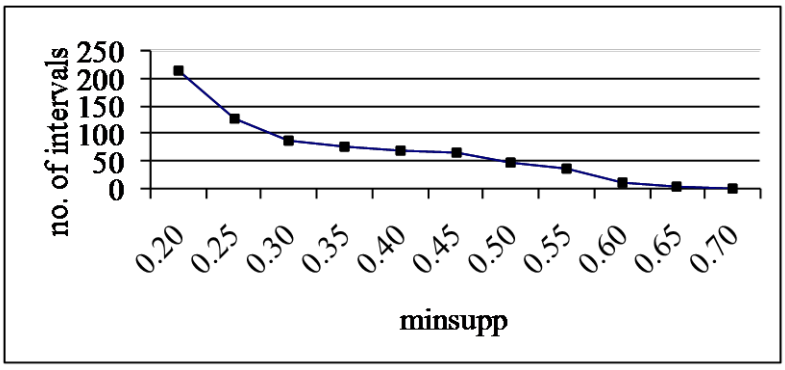

Figure 4.12 *retail* (*mininterval* = 10, *maxga*p = 12)

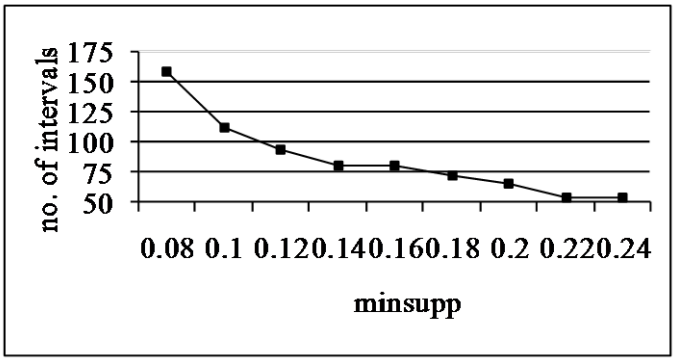

Figure 4.13 *BMS-WebView-1*(*mininterval* = 7, *maxga*p = 10)

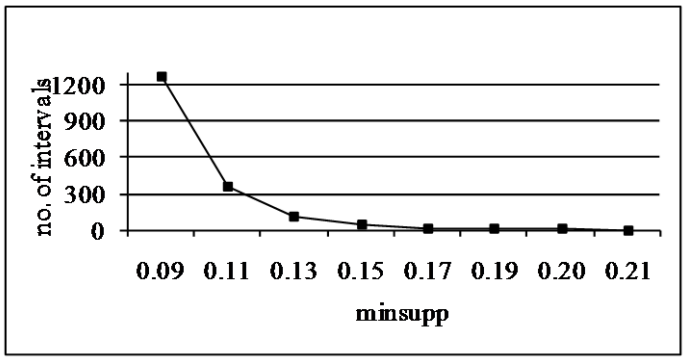

Figure 4.14. *T10I4D100K* (*mininterval* = 7, *maxga*p = 9)

4.5.3 Performance analysis

In Section 4.4.1 we have discussed the improvement on existing algorithm. We have conducted experiments to study the performance analysis based on two parameters size of the databases and *minsupp* between existing algorithm and proposed algorithm. The effect of database size and the execution time to mine the patterns using both algorithms are shown in Figures 4.15, 4.16 and 4.17. The number of patterns increases as the number of transactions increases. Thus, the execution time normally increases with the increase of size of database. In the following figures one could notice that the difference of execution time between two algorithms increases due to the increase of the size of dataset. Since we have stored the time-stamps of the transactions in an array and it takes less time to search the required time-stamp. Therefore, initially both the algorithms take almost equal time. From Figures 4.15 and 4.16 one could also observe that execution time to mine 88162 and 1,49,639 transactions of *retail* and *BMS-WebView-1* takes nearly equal time. The reason is that the average length of transaction is higher in *retail* than that of *BMS-WebView-1*. In Figure 4.16 upto transactions 72989 both the algorithms take same execution time, later the difference is observed because the *ALT* is 2 in *BMS-WebView-1*. Therefore, execution time not only depends on the size of the database but also depends on

other factors like *ALT* and *NI*. We observe that our algorithm scales linearly with the size of the database.

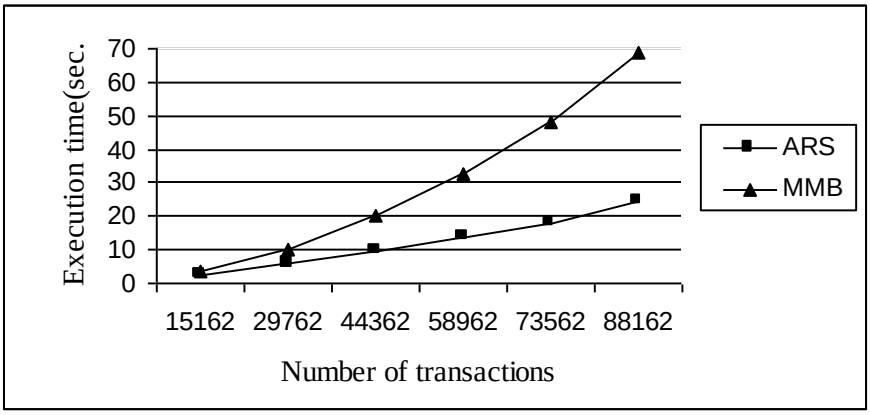

Figure 4.15 Execution time vs. size of database at *minsupp* = 0.25, *mininterval* = 8, *maxga*p = 10 (*retail*)

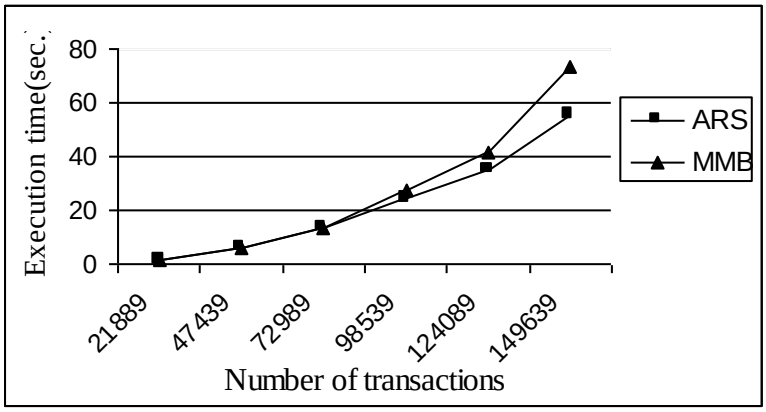

Figure 4.16 Execution time vs. size of database at *minsupp* = 0.1, *mininterval* = 7, *maxga*p = 10 (*BMS-WebView-1*)

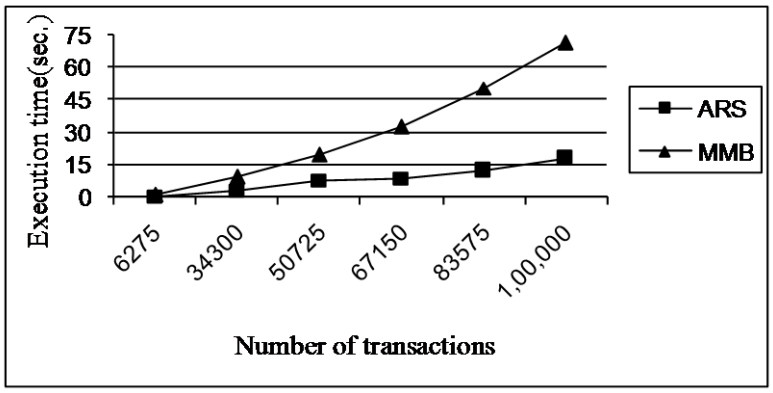

Figure 4.17 Execution time vs. size of database at *minsupp* = 0.13, *mininterval* = 7, *maxga*p = 10 (*T10I4D100K*)

In the following Figures 4.18, 4.19, and 4.20 we have presented the comparison by considering the execution time and *minsupp*. When the support increases the number of frequent itemsets decrease and so the execution time also decreases. The experimental results have shown that the execution time of both the algorithms decreases very slowly when the support threshold increases, since both the algorithms scan the database once and store the intervals of itemsets of size one. Unlike apriori it extracts the frequent itemsets of size two from the overlapped intervals. Our algorithm outperforms existing algorithm, since we store the intervals and corresponding support values in proposed hash based data structure.

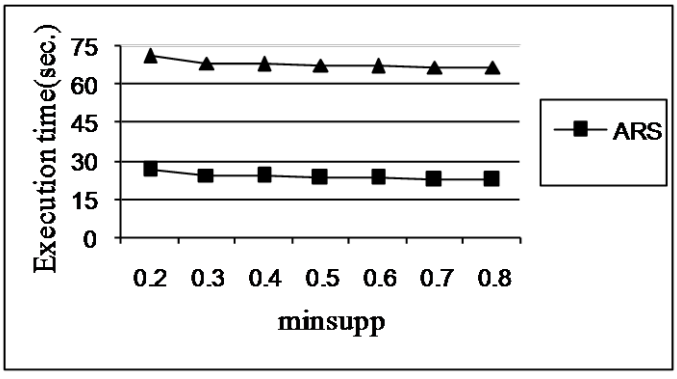

Figure 4.18 Execution time vs. *minsupp* (*mininterval* = 8, *maxga*p = 10) for *retail*

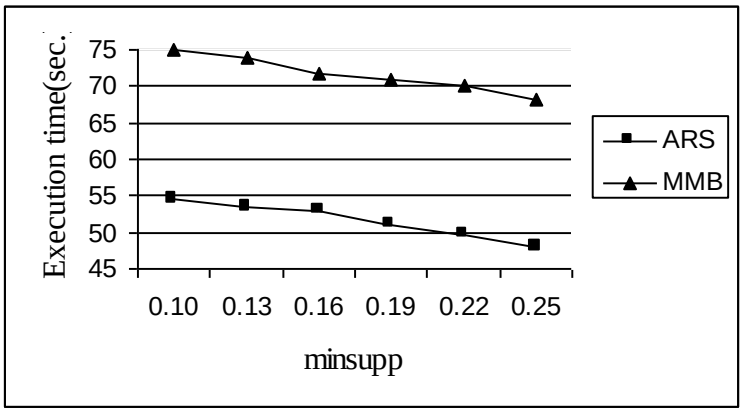

Figure 4.19 Execution time vs. *minsupp* (*mininterval* = 7, *maxga*p = 10) for *BMS-WebView-1*

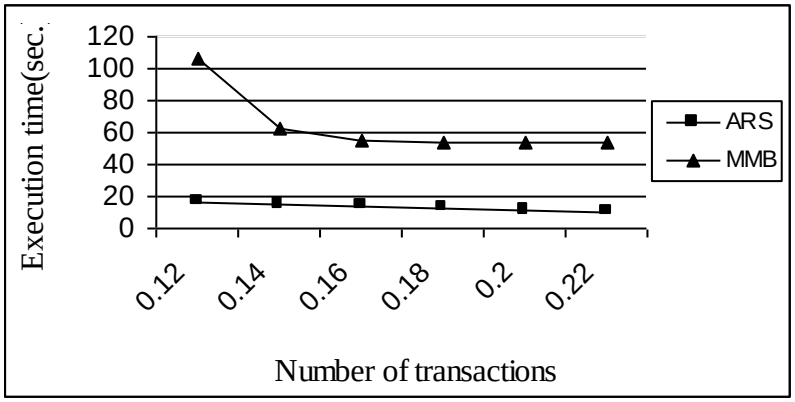

Figure 4.20 Execution time vs. *minsupp* (*mininterval* = 7, *maxga*p = 10) for *T10I4D100K*

4.6 Conclusion

In this chapter we mined locally frequent itemsets along with the set of intervals and their support range. We also extended the concept of certainty factor in association with an overlapped interval. Using this concept one can extract various calendar-based patterns viz., yearly, monthly,

weekly and daily. In addition, one could check whether any periodicity (full / partial) exists in the patterns. We have proposed some improvements in the algorithm for identifying calendar-based periodic pattern in a time-stamped dataset by introducing suitable data structure. The algorithm is incremental in nature. We have presented extensive data analysis on three data sets. We also analysed the constraints *mininterval*, *minsupp* and *maxga*p associated with each interval. In addition we have compared our algorithm with the existing algorithm. Experimental results show that the proposed algorithm runs faster than the existing algorithm.

Chapter 5

Measuring Influence of an Item in a Database Over Time

5.1 Introduction

Every time a customer interacts with a business organization there is an opportunity to gain strategic knowledge. Transactional data contains a wealth of information about customers and their purchase patterns. In fact, it could be one of the most valuable assets, when used wisely. It has been recognized by many large departmental stores, supermarkets, insurance companies, healthcare organizations, telecommunications, and banks for a long time. These organizations have spent large resources for collecting and analyzing transactional data. To stay competitive, transactional data mining is now a necessity. Many applications are based on inherent knowledge present in a database (Gary & Petersen, 2000; Wu et al., 2005; Adhikari et al., 2009). Such applications could be dealt with mining (Han et al., 2000; Agrawal & Srikant, 1994; Savasere et al., 1995) the database under consideration. In this context, many patterns e.g., frequent itemset (Agrawal et al., 1993), association rule (Agrawal et al., 1993), negative association rule (Wu et al., 2004), Boolean expression induced by itemset (Adhikari & Rao, 2007c), conditional pattern (Adhikari & Rao, 2008c) are mined / synthesized from a database. Nevertheless, there are some applications for which association-based analysis might be inappropriate. For example, an organization might deal with a large number of items with its customers. The company might be interested in knowing how the purchase of a particular item affects the purchase of another item. In this chapter, we study such influences based on transactional time stamped databases.

Many companies transact a large number of products (items) with their customers. It might be required to perform data analyzes involving different items. Such analyzes might originate from different applications. One such analysis is identifying stable items (Adhikari et al., 2009) in a database over time. It could be useful in devising strategies for a company.

Little work has been reported on data analyzes over time. In this chapter, we present another application involving different items in a database over time.

Consider a company that collects a huge amount of transactional data on yearly basis. Let DT_i be the database corresponding to the i-th year, for $i = 1, 2, \ldots, k$. Each of these databases corresponds to a specific period of time. Thus, one could call these as time databases. Each time database is mined using a traditional data mining technique (Agrawal, et al., 1993). In this application, we shall deal with the itemsets in a database. An itemset is a set of items in the database. Let I be the set of all items in the time databases. Each itemset X in a database D is associated with a statistical measure, called *support* (Agrawal, et al., 1993) denoted by $supp(X, D)$. The support of an itemset is defined as the fraction of transactions containing the itemset.

Solutions of many problems are based on the study of relationships among variables. We shall see later that the study of influence of a set of variables on another set of variables might not be the same as the association between these two sets of variables. Association analysis among variables has been studied well (Agrawal, et al., 1993; Adhikari & Rao, 2007a; Brin et al., 1997; Shapiro, 1991; Adhikari & Rao, 2008b; Adhikari & Rao, 2008c). In the context of studying association among variables using association rules one could conclude that the confidence of the association rules gives positive influence of antecedent on the consequent of the association rule. Such positive influences, though important, but might not be sufficient for many data analyses.

Consider an established company possessing data over fifty consecutive years. Generally, the sales of a product vary from one season to another season. Also, a season reappears on a yearly basis. Thus, we divide the entire database into a sequence of yearly databases. In this context, a

yearly database could be considered as a time database. In this study, we estimate the influence of an item y on x, for $y \in I$. In Section 2, we define the concept of influence of an itemset on another itemset.

An itemset could be viewed as a basic type of pattern in a database. Different types of pattern in a database could be derived from the itemset patterns. For example, frequent itemset (Agrawal et al., 1993), association rule (Agrawal et al., 1993), negative association rule (Wu et al., 2004), Boolean expression induced by itemset (Adhikari & Rao, 2007c), conditional pattern (Adhikari & Rao, 2008c) are examples of derived patterns in a database. Few applications have been reported on analysis of patterns over time. In this chapter, we wish to study the influence of an item on specific items.

Rest of the chapter is organized as follows. In Section 5.2, we extend the notion of overall association between two itemsets in a database. In Section 5.3, we introduce the notion of overall influence of an item on another item in a database. We study various properties of proposed measure. Also, we introduce the notion of overall influence of an item on a set of specific items in a database. In addition, we discuss the motivation of the proposed problem in this section. We state our problem in Section 5.4. We discuss work related to proposed problem in Section 5.5. In Section 5.6, we design an algorithm to measure the overall influence of an item on another item incrementally. In addition, we design an algorithm of overall influence of an item on a set of specific items incrementally. Experimental results are provided in Section 5.7.

5.2 Association between two itemsets

Adhikari and Rao (2007a) have proposed a measure, *OA*, of computing overall association between two items in a market basket data. The positive association between two itemsets in a

database D is defined as follows (Adhikari & Rao, 2007a) :

$PA(X, Y, D) = \frac{\#\text{transaction containing both } X \text{ and } Y \text{ in } D}{\#\text{transaction containing at least one of } X \text{ and } Y \text{ in } D}$, where X and Y are itemsets in D.

Similarly, negative association NA (Adhikari and Rao, 2007a) between two items in a database, one could be extended follows (Adhikari & Rao, 2007a):

$NA(X, Y, D) = \frac{\#\text{transaction containing exactly one of } X \text{ and } Y \text{ in } D}{\#\text{transaction containing at least one of } X \text{ and } Y \text{ in } D}$, where X and Y are itemsets in D.

Using PA and NA, OA between two itemsets X and Y in database D could be defined as follows:

$OA(X, Y, D) = PA(X, Y, D) - NA(X, Y, D)$ **... (1)**

If $OA(X, Y, D)$ is positive, negative or zero then all the items in X together and all the items in Y together are positively, negatively or independently associated in D, respectively. We illustrate different types of association in the following example.

Example 1. Let database D_1 contain the following transactions: {*a, d, e*}, {*a, b, c, d, g*}, {*a, b, e, g*}, {*b, c, g*}, {*d, e, g*}, {*b, e, f*}, {*c, d, e, f*}, {*a, b, c, d, f, g*}, and {*a, b, c, d, e*}. We find here overall association between itemsets *X*, and *Y*, for some itemsets *X*, *Y* in *D*. In Table 5.1, supports of some itemsets are given below. Here $PA(\{a, b\}, \{c, d\}, D_1) = 3/5$ and $NA(\{a, b\}, \{c, d\}, D_1) = 2/5$. Therefore, $OA(\{a, b\}, \{c, d\}, D_1) = 1/5$. In Table 5.2, overall associations are given.

In Table 5.2, we observe that the OA value between {*a, b*} and {*c, d*} as well as {*a, c*} and {*b, d*} are positive. But, the OA value between {*c*} and {*d, e*} is negative.

Table 5.1 Supports of itemsets in D_1

Itemset($\{X\}$)	$\{a, b\}$	$\{c, d\}$	$\{a, c\}$	$\{b, d\}$	$\{d, e\}$	$\{e, g\}$
$supp(\{X\}, D_1)$	4/9	4/9	3/9	3/9	4/9	2/9

Table 5.2 Overall association between two itemsets

Itemset($\{X, Y\}$)	$\{\{ab\},\{cd\}\}$	$\{\{ac\},\{bd\}\}$	$\{\{c\},\{de\}\}$
$OA(X, Y, D_1)$	1/5	1	-3/6

5.3 Concept of influence

Let X and Y be two itemsets in database D. We wish to find influence of X on Y in D. The influence of X on Y seems different from the overall association of X and Y. In the following section, we review the concept of overall association between X and Y.

Let $X = \{x_1, x_2, \ldots x_p\}$, $Y = \{y_1, y_2, \ldots y_q\}$ and $X \cap Y = \phi$ be two itemsets in database D. The influence of X on Y could be judged by the following events: (i) Whether a customer purchases all the items of Y when they purchase all the items of X i.e., the transaction contains both X and Y, and (ii) Whether a customer purchases all the items of Y when they do not purchase all the items of X i.e., the transaction contains only Y but not X. Such behaviors could be modeled using supports of $X \cap Y$ and $\neg X \cap Y$. The expression $supp(X \cap Y, D)/supp(X, D)$ measures the strength of positive association of X on Y. The expression $supp(\neg X \cap Y, D)/supp(\neg X, D)$ measures the strength of negative association of X on Y. Thus, the expressions $supp(X \cap Y, D)/supp(X, D)$ and $supp(\neg X \cap Y, D)/supp(\neg X, D)$ could be important in measuring overall influence of X on Y.

5.3.1 Influence of an itemset on another itemset

Let X and Y be the two itemsets in database D. The interestingness of an association rule r_1: $X \rightarrow Y$ could be expressed by its support and confidence (*conf*) measures (Agrawal et al., 1993). These measures are defined as follows. $supp(r_1, D) = supp(X \cap Y, D)$, and $conf(r_1, D) = supp(X \cap Y, D) / supp(X, D)$. $conf(r_1, D)$ could be interpreted as the fraction of transactions containing itemset Y among the transactions containing X in D. In other words, $conf(r_1, D)$ could be viewed as the *positive influence* (*PI*) of X on the itemset Y. Let us consider the negative association rule r_2: $\neg X \rightarrow Y$. Confidence of r_2 in D could be viewed as fractions of transactions containing Y among the transactions containing $\neg X$. In other words, confidence of r_2 in D could be viewed as *negative influence* (*NI*) of X on Y. In similar to overall association defined in (1), one could define *overall influence* (*OI*) of X on Y in a database as follows.

Definition 1. Let X and Y be two itemsets in database D such that $X \cap Y = \phi$. Then overall influence of X on Y in D is defined as follows:

$$OI(X, Y, D) = supp(X \cap Y, D)/supp(X, D) - supp(\neg X \cap Y, D)/supp(\neg X, D) \quad \textbf{...(2)}$$

$OI(X, Y, D)$ represents the difference of the influence on Y when X is present in a transaction and the influence on Y when X is not present in the transaction. Let γ be user-defined level of interestingness. Then $OI(X, Y, D)$ is interesting if $OI(X, Y, D) \geq \gamma$.

If $OI(X, Y, D) > 0$ then the itemset X has positive influence on itemset Y in D. In other words, all the items in X together help promoting itemset Y in D. If $OI(X, Y, D) < 0$ then X has negative influence on Y in D. In other words, all the items in X in D together do not help promoting

together all the items in Y. If $OI(X, Y, D) = 0$ then X has no influence on Y in D.

Let $X = \{bread, butter\}$, $Y = \{milk, sugar\}$ and $X \cap Y = \phi$ be two itemsets in database D. Then one can compute the $OI(X, Y, D)$ by using equation (2).

In Example 2, we illustrate the concept of overall influence.

Example 2. We continue our discussion with reference to Example 1. We have $PI(\{a, b\}, \{c, d\}, D_1) = 3/4$, $NI(\{a, b\}, \{c, d\}, D_1) = 1/5$, and $OI(\{a, b\}, \{c, d\}, D_1) = 11/20$. In D_1, given itemset $\{a, b\}$, itemset $\{c, d\}$ occurs frequently. We observe that $PI(\{a, b\}, \{c, d\}, D_1)$ is more than $PA(\{a, b\}, \{c, d\}, D_1)$. Also, $NA(\{a, b\}, \{c, d\}, D_1)$ is more than $NI(\{a, b\}, \{c, d\}, D_1)$. So, $OI(\{a, b\}, \{c, d\}, D_1)$ is more than $OA(\{a, b\}, \{c, d\}, D_1)$. In similar to overall association, overall influence could be negative also. Let $X = \{c\}$ and $Y = \{d, e\}$. $PI(X, Y, D_1) = 2/5$, $NI(X, Y, D_1) = 1/2$, and $OI(X, Y, D_1) = -1/10$. Thus, overall influence between two itemsets could be negative as well as positive.

In most of the cases, the value of overall influence between two itemsets in a large database is negative. In real databases, it might be possible that the overall influence between the two itemsets is positive. In Example 3, we consider some special cases to illustrate the measure of overall influence.

Example 3. Let database D_2 contains following transactions: $\{a, b, e\}$, $\{a, e, g\}$, $\{b, e, g\}$, $\{a, b, d, e, g\}$, $\{b, d, e, g\}$ and $\{c, e, g\}$. We compute overall influence of an itemset X on another itemset Y under various cases.

Case 1: $supp(X, D_2) > supp(Y, D_2)$
Let $X = \{e, g\}$, $Y = \{a, b\}$. $supp(X, D_2) = 5/6$, $supp(Y, D_2) = 2/6$ and $supp(X \cap Y, D_2) = 1/6$. We get $OI(X, Y, D_2) = -0.8$.
Case 2: $supp(X, D_2) < supp(Y, D_2)$

Let $X = \{a, b\}$, $Y = \{e, g\}$. $supp(X, D_2) = 2/6$, $supp(Y, D_2) = 5/6$ and $supp(X \cap Y, D_2) = 1/6$. We get $OI(X, Y, D_2) = -0.5$.

Though the values of overall influence are negative for the above cases, the influence might turn positive for some databases. Let us consider another database $D_3 = \{\{a, b, c, d, g\}, \{b, c, g\}, \{c, d, g\}, \{a, b, c, d, e\}, \{b, c, e, g\}, \{a, b, c, d,$ e, $g\}$,

Case 1: $supp(X, D_3) > supp(Y, D_3)$

Let $X = \{c, d\}$, $Y = \{a, b\}$. $supp(X, D_3) = 4/6$, $supp(Y, D_3) = 3/6$ and $supp(X \cap Y, D_3) = 3/6$. We get $OI(X, Y, D_3) = 0.5$.

Case 2: $supp(X, D_3) < supp(Y, D_3)$. Let $X = \{a, b\}$, $Y = \{c, d\}$. $supp(X, D_3) = 3/6$, $supp(Y, D_3) = 4/6$ and $supp(X \cap Y, D_3) = 3/6$. We get $OI(X, Y, D_3) = 0.667$.

5.3.2 Properties of influence measures

For the purpose of computing influence of an itemset on another itemset, one needs to express *OI* in terms of supports of relevant itemsets. From (2), we get *OI* as follows:

$OI(X, Y, D) = supp(X \cap Y, D)/supp(X, D) - (supp(Y, D) - supp(X \cap Y, D)) / (1 - supp(X, D)$

Finally, we get *OI* as follows:

$$OI(X, Y, D) = \frac{supp(X \cap Y) - supp(X) \times supp(Y)}{supp(X)[1 - supp(X)]}, \text{ if } supp(X, D) \neq 1 \text{ or } supp(Y, D) \neq 1$$

$OI(X, Y, D) = 0$, otherwise **... (3)**

From the above formula one could observe that if support of itemset X in D is 1 then influence of other itemsets on X will be zero. On the other hand, if $supp(Y, D) = 1$ then $supp(X \cap Y, D) = supp(X, D)$ and $supp(X, D) \times supp(Y, D) = supp(X, D)$. Therefore, the numerator of formula (3) will result in zero and overall influence becomes zero. In the following proposition, we state

some properties of *PI* and *NI*. *OI*(*X, X, D*) = 1 at *X* = *Y*. Thus, *OI*(*X, X, D*) at *X* = *Y* could be termed as *trivial influence*.

Proposition 1. For itemsets *X*, *Y* in *D*, the following properties are satisfied: (i) $0 \le PI(X, Y, D) \le 1$, (ii) $0 \le NI(X, Y, D) \le 1$, (iii) $-1 \le OI(X, Y, D) \le 1$.

Proposition 2. $OI(X, Y, D) = \frac{\text{supp}(Y)[Corr(X,Y,D)-1]}{1-\text{supp}(X)}$, where *Corr*(*X, Y, D*) is the correlation coefficient between itemsets *X* and *Y* in database *D*.

Proof. From equation (3) we get $OI(X, Y, D) = \frac{supp(X \cap Y) - supp(X) \times \text{supp}(Y)}{supp(X)[1-supp(X)]}$ and

$Corr(X, Y, D) = \frac{supp(X \cap Y)}{supp(X) \times supp(Y)}$

Therefore, $OI(X, Y, D) = \frac{supp(X) \times supp(Y) \times Corr(X,Y,D) - supp(X) \times supp(Y)}{supp(X)[1-supp(X)]} =$

$\frac{supp(X) \times supp(Y)[Corr(X,Y,D)-1]}{supp(X)[1-supp(X)]} = \frac{supp(Y)[Corr(X,Y,D)-1]}{[1-supp(X)]}$

If *Corr*(*X, Y, D*) = 1 then *X* and *Y* are independent in database *D*. In other words, if *OI*(*X, Y, D*) = 0 then *X* and *Y* are independent in *D*. If *Corr*(*X, Y, D*) < 1 then *X* and *Y* are negatively correlated in database *D*. In other words, if *OI*(*X, Y, D*) < 0 then *X* and *Y* are negatively correlated. If *Corr*(*X, Y, D*) > 1 then *X* and *Y* are positively correlated in database *D*. In other words, if *OI*(*X, Y, D*) > 0 then *X* and *Y* are positively correlated.

5.3.3 Influence of an item on a set of specific items

Let $I = \{i_1, i_2, \ldots, i_m\}$ be the set of items in database *D*. Also, let $SI = \{s_1, s_2, \ldots, s_p\}$ be the set of specific items in database *D*. We would like to study the overall influence of each item on *SI*.

The influence of an item on *SI* could be computed based on $OI(i_j, s_k, D)$, for $j = 1, 2,\ldots, m$ and $k = 1, 2,\ldots, p$. Let γ be the user-defined *minimum influence level*. Thus, the influence of i_j on s_k is interesting if $OI(i_j, s_k, D) \geq \gamma$, for $j = 1, 2,\ldots, m$ and $k = 1, 2,\ldots, p$. The procedure of determining influence of an item on a set of specific items could be explained using the following steps.

(i) Generate influence matrix (*IM*) of order $p \times n$ using $OI(i_j, s_k, D)$, for $j = 1, 2,\ldots, m$ and $k = 1, 2,\ldots, p$. (ii) An influence is counted when it is interesting. (iii) For each item, count the number of interesting influences on each of the specific items. (iv)The items in database *D* are sorted based on primary key as the number of interesting influences on the specific items, and secondary key as the support of an item. We explain steps (i)-(iv) using Example 3.

Example 4. Consider the database D_1 given in Example 1. Let $I = \{a, b, c, d, e, f, g\}$ and $SI = \{a, c, d\}$.

Table 5.3 Supports of each items in D_1

Items (x)	*a*	*b*	*c*	*d*	*e*	*f*	*g*
$supp(\{x\}, D_1)$	5/9	6/9	5/9	6/9	6/9	3/9	5/9

In this case, the influence matrix is of order 3×7 as given below.

$$IM = \begin{array}{c|ccccccc} item & a & b & c & d & e & f & g \\ a & 1 & 0.3333 & 0.1 & 0.3333 & -0.1667 & -0.3333 & 0.35 \\ c & 0.1 & 0.3333 & 1 & 0.3333 & -0.6666 & 0.1667 & 0.35 \\ d & 0.3 & -0.5 & 0.3 & 1 & 0 & 0 & -0.15 \end{array}$$

Let γ be 0.2. Also let $x(\eta)$ denote η number of interesting influences of item x on specific items. The interesting influences of different items in D_1 are given as follows. $a(2)$, $b(2)$, $c(2)$, $d(3)$, $e(0)$, $f(0)$, $g(2)$. The items are sorted using step (iv), and they are given as follows. $d(3)$, $b(2)$, $a(2)$, $c(2)$, $g(2)$, $e(0)$, $f(0)$. Given the set of specific items $\{a, c, d\}$, one could conclude that the item d has the maximum and the item f has a minimum influence on the set $\{a, c, d\}$.

5.3.4 Motivation

The concept of influence might not be new in the literature of data mining. For example, $conf(X \rightarrow Y, D)$ refers to positive influence of X on Y. In other words, it implies how likely a customer purchases the items of Y when the customer has already purchased all the items of X. In addition, the concept of negative influence existed in the literature of data mining. $conf(\neg X \rightarrow Y, D)$ refers to the amount of negative influence of items of X in purchasing the items of Y. In other words, it implies how likely a customer purchases the items of Y when the customer has not purchased all the items of X. In many data analyzes it might be required to consider the overall influence of a set of items on another set of items. Our work introduces the notion of overall influence that could be useful in dealing with many real life problems. In the following paragraph, we justify that no existing measure might be appropriate to study the overall influence of an itemset on another itemset.

The analysis of relationships among variables is a fundamental task being at the heart of many data mining problems. For example, metrics such as support, confidence, lift, correlation, and collective strength have been used extensively to evaluate the interestingness of association patterns. These metrics are defined in terms of the frequency counts tabulated in a 2×2

contingency table as shown in Table 5.4. To illustrate this, consider the ten example contingency tables, E_1 to E_{10}, given in Table 5.5. Tan et al. (2003) present an overview of twenty one interestingness measures proposed in the statistics, machine learning and data mining literature.

In the following discussion, we shall observe why these measures fail to compute overall influence of an itemset on another itemset. In Examples 2 and 3, we have observed that the overall influence of an itemset on another itemset could be positive as well as negative. Thus, overall influence of an itemset on another itemset in a database lies in [-1, 1]. In a large database, where items are sparsely distributed over the transactions might result in negative overall influence of an itemset on another itemset. Based on these observations, one could consider the following five out of twenty one interestingness measures since overall influence of an itemset on another itemset lies in [-1, 1]. These measures are presented in Table 5.6.

Table 5.4 A 2 × 2 contingency table for variables X and Y

	Y	$\neg Y$	Total
X	f_{11}	f_{10}	f_{1+}
$\neg X$	f_{01}	f_{00}	f_{0+}
Total	$f_{.+1}$	$f_{.+0}$	N

Table 5.5 Examples of contingency tables

Example	f_{11}	f_{10}	f_{01}	f_{00}
$E1$	8123	83	424	1370
$E2$	8330	2	622	1046
$E3$	9481	94	127	298
$E4$	3954	3080	5	2961
$E5$	2886	1363	1320	4431
$E6$	1500	2000	500	6000
$E7$	4000	2000	1000	3000
$E8$	4000	2000	2000	2000
$E9$	1720	7121	5	1154
$E10$	61	2483	4	7452

Table 5.6 Relevant interestingness measures for association patterns

Symbol	Measure	Formula
ϕ	ϕ-coefficient	$\frac{P(\{x\}\cup\{y\})-P(\{x\})\times P(\{y\})}{\sqrt{P(\{x\})\times P(\{y\})\times(1-P(\{x\})\times(1-P(\{y\}))}}$
Q	Yule's Q	$\frac{P(\{x\}\cup\{y\})\times P(\neg(\{x\}\cap\{y\}))-P(\{x\}\cup\neg\{y\})\times P(\neg\{x\}\cup\{y\})}{P(\{x\}\cup\{y\})\times P(\neg(\{x\}\cap\{y\}))-P(\{x\}\cup\neg\{y\})\times P(\neg\{x\}\cup\{y\})}$
Y	Yule's Y	$\frac{\sqrt{P(\{x\}\cup\{y\})\times P(\neg(\{x\}\cap\{y\}))}-\sqrt{P(\{x\}\cup\neg\{y\})\times P(\neg\{x\}\cup\{y\})}}{\sqrt{P(\{x\}\cup\{y\})\times P(\neg(\{x\}\cap\{y\}))}-\sqrt{P(\{x\}\cup\neg\{y\})\times P(\neg\{x\}\cup\{y\})}}$
κ	Cohen's	$\frac{P(\{x\}\cup\{y\})+P(\neg\{x\}\cup\neg\{y\})-P(\{x\})\times P(\{y\})-P(\neg\{x\})\times P(\neg\{y\})}{1-P(\{x\})\times P(\{y\})-P(\neg\{x\})\times P(\neg\{y\})}$
F	Certainty factor	$\max\left(\frac{P(\{y\}\|\{x\})-P(\{y\})}{1-P(\{y\})},\frac{P(\{x\}\|\{y\})-P(\{x\})}{1-P(\{x\})}\right)$

In Table 5.7, we rank the contingency tables using each of the above measures under consideration.

Table 5.7 Ranking of contingency tables using above interestingness measures

Example	ϕ	Q	Y	κ	F
$E1$	1	3	3	1	4
$E2$	2	1	1	2	1
$E3$	3	4	4	3	6
$E4$	4	2	2	5	2
$E5$	5	8	8	4	9
$E6$	6	7	7	7	7
$E7$	7	9	9	6	8
$E8$	8	10	10	8	10
$E9$	9	5	5	9	3
$E10$	10	6	6	10	5

Also, we rank the contingency tables based on the concept of overall influence explained in Example 1. In Table 5.8, we present the ranking of contingency tables using overall influence.

Table 5.8 Ranking of contingency tables using overall influence

Example	Overall influence	Rank
$E1$	0.754	1
$E2$	0.627	3
$E3$	0.691	2
$E4$	0.560	4
$E5$	0.450	5
$E6$	0.352	7
$E7$	0.417	6
$E8$	0.167	9
$E9$	0.190	8
$E10$	0.023	10

None of the five measures ranks contingency tables like the ranks given in Table 5.7. Thus, none of the above five measures serves the special requirement of the proposed problem.

5.4 Problem statement

Let *D* be a database of customer transactions grown over a period of time. In this chapter, we are interested in making influence analysis of a set of specific items. We will see how each of the specific items gets influenced by different items in the database. As the database grows over time, an incremental solution to influence analysis of specific items is a natural and desirable solution. To provide an incremental solution to this problem, one might require sequence of databases over time. Each time database corresponds to the set of transactions made for a specific period of time. In this regard, the choice of time period corresponding to a database is an important issue. One could observe that the sales of items might vary over different seasons in a year. Instead of processing all the data together, we process data on yearly basis. Then, the result of processing for the current year could be combined with that of previous years. Such incremental analysis might be appropriate since a season reappears on a yearly basis. Otherwise, processed result might be biased due to seasonal variations.

The goal of this chapter is to make an influence analysis of a set of items in a database. Let D_t be the database for the t-th period of time, $t = 1, 2, \ldots, n$. For computing overall influence between two items in a database, one needs to mine supports of itemsets of size 1 and size 2. The *size* of an itemset refers to the number of items in the itemset. Let $D_{1,k}$ be the collection of databases $D_1, D_2, \ldots, D_k$. For computing $OI(x, y, D_{1,k+1})$, we assume that $OI(x, y, D_{1,k})$ is available to us for items x, y in the given database. In other words, for computing $OI(x, y, D_{1,k+1})$, we have

supp(x, $D_{1,k}$), *supp*(y, $D_{1,k}$), and *supp*($x \cap y$, $D_{1,k}$). Thus, our incremental procedure needs to compute *supp*(x, $D_{1,k+1}$), *supp*(y, $D_{1,k+1}$), and *supp*($x \cap y$, $D_{1,k+1}$) using (i) *supp*(x, $D_{1,k}$), *supp*(y, $D_{1,k}$), and *supp*($x \cap y$, $D_{1,k}$), (ii) *supp*(x, D_{k+1}), *supp*(y, D_{k+1}), and *supp*($x \cap y$, D_{k+1}). In general, for an itemset X in the database, *supp*(X, $D_{1,k+1}$) could be obtained incrementally as follows.

$$supp(X, D_{1,k+1}) = \frac{size(D_{k+1}) \times supp(X, D_{k+1}) + size(D_{1,k}) \times supp(X, D_{1,k})}{size(D_{k+1}) + size(D_{1,k})} \quad \textbf{... (4)}$$

The *size*(D) refers to the number of transactions in database D.

5.5 Related work

In analyzing positive association between itemsets in a database, support-confidence framework was established by Agarwal et al. (1993). In Section 5.2.3, we have discussed why confidence measure alone is not sufficient in determining overall influence of an itemset on another itemset. Also, interestingness measures such as support, collective strength (Aggarwal & Yu, 1998) and Jaccard (Tan et al., 2003) are not relevant in this context, since they are 1-argument measures.

The χ^2 test (Greenwood & Nikulin, 1996) only tells us whether two or more items are dependent. Such a test answers either "yes" or "no" to the question of whether the association is meaningful, and hence it might not be suitable for the specific requirement of our problem.

The interestingness measures such as lift (Tan et al., 2003), correlation (Tan et al., 2003), conviction (Brin et al., 1997), and odds-ratio (Tan et al., 2003) are semantically different from the measure of overall influence. Moreover, each of these measures lies in [0, ∞).

Shapiro (1991) has proposed leverage measure in the context of mining strong rules in a database. It might not be suitable for the specific requirement of our problem.

5.6 Design of Algorithms

Based on the discussion held in previous section, we design three algorithms for measuring influence of an item on another item and influence of an item on a set of specific items.

5.6.1 Designing Algorithm for measuring overall influence of an item on another item

In this algorithm, we measure influence of an item on each the items incrementally. We have expressed influence of an itemset on another itemset using supports of the relevant itemsets.

Each itemset could be described by its *itemset* and *support*. We maintain *IS1* and *IS2* for storing itemsets in $D_{1,k}$ of size one and two, respectively. *Itemset* attribute of *i*-th 1-itemset could be accessed using the notation *IS1*[*i*].*itemset*. Similar notation is used to access *support* attribute of an itemset. Also, we maintain Δ*IS1* and Δ*IS2* for storing itemsets in D_{k+1} of size one and two, respectively. We merge *IS1*and Δ*IS1* to obtain supports of 1-itemsets in $D_{1,k+1}$ and are stored in array *OIS1*. Similarly, we merge *IS2* and Δ*IS2* to obtain supports of 2-itemsets in $D_{1,k+1}$ and are stored in array *OIS2*. Using *OIS1* and *OIS2*, we compute overall influence between items in $D_{1,k+1}$. Overall influence between items is computed using formula (3) and is stored in array *IOI*. The overall influence corresponding to *i*-th pair of items is accessed by *IOI* [*i*].*oi*.

Algorithm 1. Find top *k* overall influences in the database over time.

procedure *Top-k-OI*(*k*, *IS1*, *IS2*, Δ*IS1*, Δ*IS2*, *IOI*)

Inputs:

k: an integer representing the number of top influences

IS1: array of supports of itemsets of size one in $D_{1,k}$

IS2: array of supports of itemsets of size two in $D_{1,k}$

Δ*IS1*: array of supports of itemsets of size one in D_{k+1}

Δ*IS2*: array of supports itemsets of size two in D_{k+1}

Outputs:

IOI: array of overall influences in $D_{1,k+1}$

01: sort array Δ*IS1* on *itemset* attribute;

02: sort array Δ*IS2* on *itemset* attribute;

03: call *Merge* (*IS1*, Δ*IS1*, *OIS1*);

04: call *Merge* (*IS2*, Δ*IS2*, *OIS2*);

05: **let** *j* = 1;

06: **for** *i* = 1 to |*OIS2*| **do**

07: search *OIS2*[*i*].*item1* in *OIS1*;

08: search *OIS2*[*i*].*item2* in *OIS1*;

09: *IOI* [*j*].*oi* = *OI*(*OIS2*[*i*].*item1*, *OIS2*[*i*].*item2*, *D*);

10: *IOI* [*j*].*item1* = *OIS2*[*i*].*item1*; *IOI* [*j*].*item2* = *OIS2*[*i*].*item2*;

11: increase *j* by 1;

12: *IOI* [*j*].*oi* = *OI*(*OIS2*[*i*].*item2*, *OIS2*[*i*].*item1*, *D*);

13: *IOI* [*j*].*item1* = *OIS2*[*i*].*item2*; *IOI* [*j*].*item2* = *OIS2*[*i*].*item1*;

14: increase *j* by 1;

15: **end for**

16: sort array *IOI* in descending order on *oi* attribute;

17: return first *k* influences;

18: **end procedure**;

Procedure *Merge* (*A*, *B*, *C*) merges sorted arrays *A* and *B* and generates output array *C*. In this context, sorting is based on support of an itemset. The time complexity of *Merge* procedure is $O(|A| + |B|)$ (Knuth, 1998). Now, *OIS1* contains the supports of items in $D_{1,k+1}$. Also, *OIS2* contains the supports of itemsets of size two in $D_{1,k+1}$.

The information contained in *OIS1* and *OIS2* is used to compute overall influence of an item on another item in $D_{1,k+1}$. In lines 06-12, we have computed influence of an item on another item in $D_{1,k+1}$. In line 16, for each item, we have sorted overall influences of different items on the amount of influence. Finally, we display first *k* items and their influences for each item.

Let *IS1* and *IS2* contain *M* and *N* itemsets respectively. Let Δ*IS1* and Δ*IS2* contain *m* and *n* elements respectively. Lines 1 and 2 take $O(m \times log(m))$ and $O(n \times log(n))$ time respectively. Also, lines 3 and 4 take $O(M + m)$ and $O(N + n)$ time respectively. Each of the search statements in lines 7 and 8 take $O(log(M + m))$ time, since *OIS1* is sorted. The sort statement in line 16 takes time $O((N + n) \times log(N + n))$. The time complexity of lines 6-15 is $O((N + n) \times log(M + m))$. The time complexity of algorithm *Top-k-OI* is *maximum* $\{O(M + m), O((N + n) \times log(N + n)), O((N + n) \times log(M + m))\}$.

5.6.2 Designing Algorithm for measuring overall influence of an item on each of the specific items

One could store specific items in an array. The proposed algorithm seems to be the same as Algorithm 1 except that every time it measures an overall influence of an item on a specific item.

5.6.3 Designing Algorithm for identifying top influential items on a set of specific items

In Algorithm 2, we find influence of an item on a set of specific items in a database. We construct influence matrix (*IM*) from the array of specific items (*SI*) and from the array of overall influence between items (*IOI*). The algorithm scans *IM* for each item to count the number of interesting influences. The number of interesting influences for each item is stored in array *count*. Finally, we sort *count* on descending order on primary key *count* and secondary key *supp*.

Algorithm 2. Find influence of an item on a set of specific items in the database over time.

procedure *Top-k-items*(*SI*, *IS1*, *IS2*, Δ*IS1*, Δ*IS2*, *OIS1*, *OIS2*, *IOI*)

Inputs:

SI: array of specific items

IS1, *IS2*, Δ*IS1*, Δ*IS2*, *OIS1*, *OIS2*, *IOI*: as specified in Algorithm 1

Outputs:

count: array of number of interesting influences

```
01:  let k = 1;
02:  for i = 1 to |SI| do
03:     for j = 1 to | IOI | do
04:        if (SI[i] = IOI [j].item1) then
05:           IM[i][j] = IOI [j].oi;
06:        end if
07:     end for
08:  end for
09:  for j = 1 to | IOI | do
```

10: **let** *count* [*j*] = 0;

11: **for** *i* = 1 to |*SI*| **do**

12: **if** (*IM*[*j*][*i*] ≥ *γ*) **then**

13: increase *count* [*j*] by 1;

14: **end if**

15: **end for**

16: **end for**

17: sort *count* on descending order on primary key count value and secondary key support;

18: return first *k* items;

end procedure;

Let array *SI* contains *p* items. Line 2 repeats for *p* times. Line 3 repeats $O(M + m)$ times. So, lines 2-8 take $O(p \times (M + m))$ time. Line 9 repeats $O(M + m)$ times. Line 11 repeats *p* times. Thus, line 9-16 take $O(p \times (M + m))$ time. Therefore, the time complexity of the above algorithm is $O(p \times (M + m))$, where $M > m$. Also, sorting statement at line 17 takes $O((M + m) \times log(M + m))$. The time complexity of algorithm top *k* items is $maximum\{O(p \times (M + m)), O((M + m) \times log(M + m))\}$.

5.7 Experiments

We have carried out several experiments to study the effectiveness of the proposed analysis. All the experiments have been implemented on a 1.6 GHz Pentium IV with 256 MB of memory using visual C++ (version 6.0) software. We present the experimental results using two real datasets and one synthetic dataset. The datasets *mushroom*, *retail* (Frequent itemset mining

dataset repository) and *ecoli* are real. Dataset *ecoli* is a subset of *ecoli database* (UCI ML repository) and it has been processed for the purpose of conducting experiments. The synthetic dataset *random-68* has been generated for the purpose of conducting experiments. The details of these datasets are given in Table 5.9.

Table 5.9 Dataset characteristics

Database	*NT*	*ALT*	*AFI*	*NI*
mushroom (*M*)	8124	24.000	1624.800	120
ecoli (*E*)	336	7.000	25.835	91
random-68 (*R*)	3000	5.460	280.985	68
retail (*Rt*)	88,162	11.306	60.54	16,470

The symbols used in different tables are explained as follows. Let *D*, *NT*, *ALT*, *AFI*, and *NI* denote database, the number of transactions, average length of a transaction, average frequency of an item, and number of items respectively. Each dataset has been divided into 10 databases, called input databases, for the purpose of conducting experiments on multiple time databases. The input databases obtained from *mushroom*, *ecoli*, *random-68* and *retail* are named as M_i, E_i, R_i, and Rt_i for $i = 0, 1, \ldots, 9$. We present some characteristics of the input databases in Table 5.10. Top 10 overall influences in different databases are shown in Table 5.11.

Table 5.10 Time database characteristics

D	*NT*	*ALT*	*AFI*	*NI*	*D*	*NT*	*ALT*	*AFI*	*NI*
M_0	812	24.000	295.273	66	M_5	812	24.000	221.454	88
M_1	812	24.000	286.588	68	M_6	812	24.000	216.533	90
M_2	812	24.000	249.846	78	M_7	812	24.000	191.059	102
M_3	812	24.000	282.435	69	M_8	812	24.000	229.271	85
M_4	812	24.000	259.840	75	M_9	816	24.000	227.721	86
E_0	33	7.000	4.620	50	E_5	33	7.000	3.915	59
E_1	33	7.000	5.133	45	E_6	33	7.000	3.500	66
E_2	33	7.000	5.500	42	E_7	33	7.000	3.915	59
E_3	33	7.000	4.813	48	E_8	33	7.000	3.397	68
E_4	33	7.000	3.397	68	E_9	39	7.000	4.550	60
R_0	300	5.590	28.676	68	R_5	300	5.140	26.676	68
R_1	300	5.417	28.000	68	R_6	300	5.510	28.353	68
R_2	300	5.360	27.647	68	R_7	300	5.497	28.338	68
R_3	300	5.543	28.456	68	R_8	300	5.537	28.471	68
R_4	300	5.533	28.382	68	R_9	300	5.477	28.235	68
Rt_0	9000	11.244	12.070	8384	Rt_5	9000	10.856	16.710	5847
Rt_1	9000	11.209	12.265	8225	Rt_6	9000	11.200	17.416	5788
Rt_2	9000	11.337	14.597	6990	Rt_7	9000	11.155	17.346	5788
Rt_3	9000	11.490	16.663	6206	Rt_8	9000	11.997	18.690	5777
Rt_4	9000	10.957	16.039	6148	Rt_9	7162	11.692	15.348	5456

Table 5.11 Top 10 overall influences in different databases

M (*supp* = 0.15)			*E* (*supp* = 0.12)			*R* (*supp* = 0.03)			*Rt* (*supp* = 0.12)		
{*x*}	{*y*}	*OI*	{*x*}	{*y*}	*OI*	{*x*}	{*y*}	*OI*	{*x*}	{*y*}	*OI*
86	34	0.997	24	48	0.946	19	29	-0.017	41	39	0.200
34	86	0.992	89	50	0.913	29	19	-0.020	39	48	0.180
58	24	0.991	53	48	0.693	8	56	-0.023	48	39	0.175
67	76	0.986	63	50	0.665	56	8	-0.023	41	48	0.129
76	67	0.986	87	50	0.660	15	14	-0.031	39	41	0.114
24	58	0.963	56	50	0.621	14	15	-0.032	48	41	0.071
93	59	0.895	61	50	0.618	18	52	-0.035	48	7	-0.234
93	76	0.884	27	48	0.618	52	18	-0.036	39	7	-0.292
93	67	0.881	83	50	0.540	54	58	-0.044	48	2	-0.293
102	24	0.875	56	48	0.488	58	54	-0.047	48	1	-0.316

We have studied execution time with respect to number of data sources. We observe in Figures 5.1, 5.2, 5.3 and 5.4 that the execution time increases as the number of data sources increases.

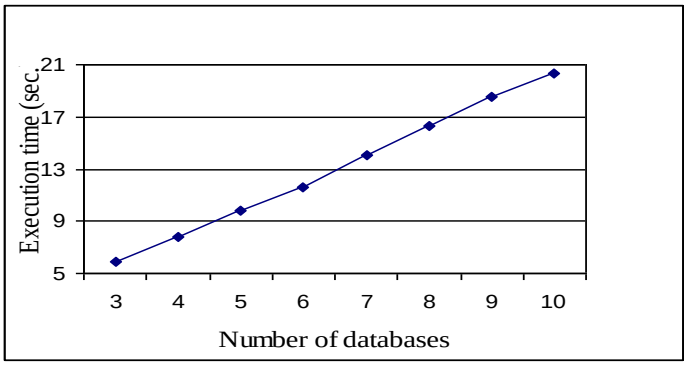

Figure 5.1 Execution time versus number of databases obtained from *mushroom* (*supp* = 0.2)

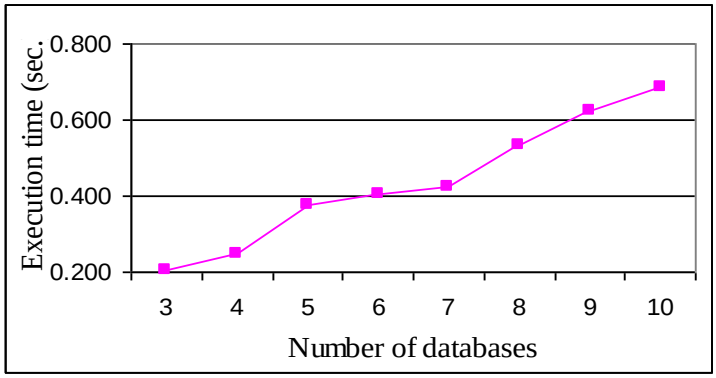

Figure 5.2 Execution time versus number of databases obtained from *ecoli* (*supp* = 0.12)

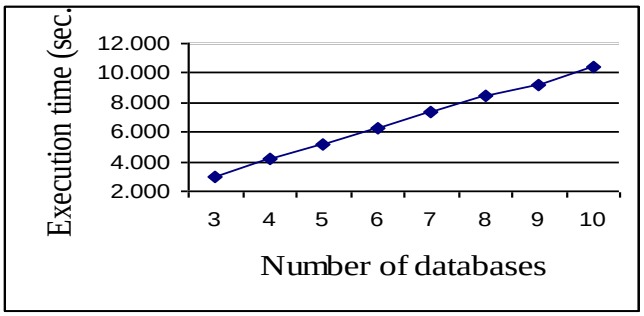

Figure 5.3 Execution time versus number of databases obtained from *random-68* (*supp* = 0.03)

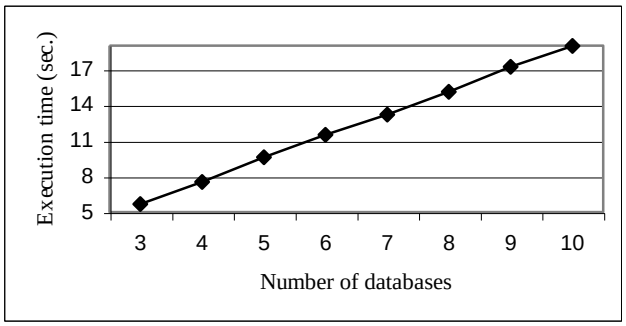

Figure 5.4 Execution time versus number of databases at *supp* = 0.2 (*retail*)

The size of each input database generated from *mushroom* and *retail* are significantly larger than an input database generated from *ecoli*. As a result, we observe a steeper graph in Figure 5.1 and 5.4. The number of frequent itemsets decreases as the minimum support increases.

In Figures 5.5, 5.6, 5.7 and 5.8 we have shown how the execution time decreases over the increase of the minimum support value.

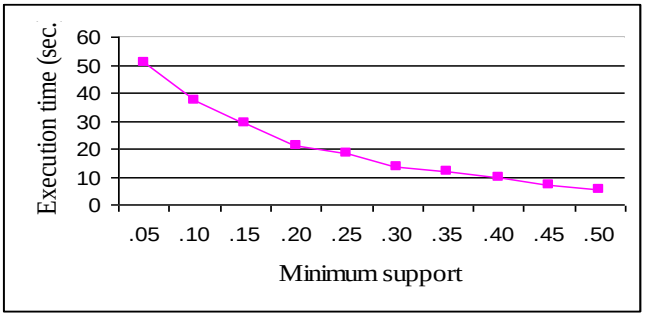

Figure 5.5 Execution time versus minimum support (for *mushroom*)

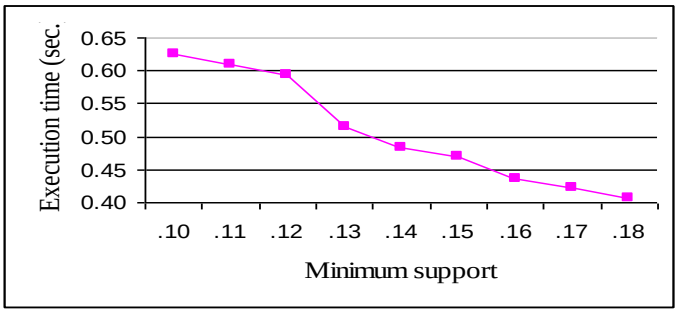

Figure 5.6 Execution time versus minimum support (for *ecoli*)

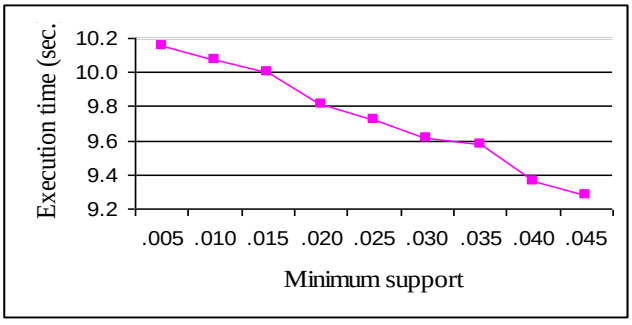

Figure 5.7 Execution time versus minimum support (for *random-68*)

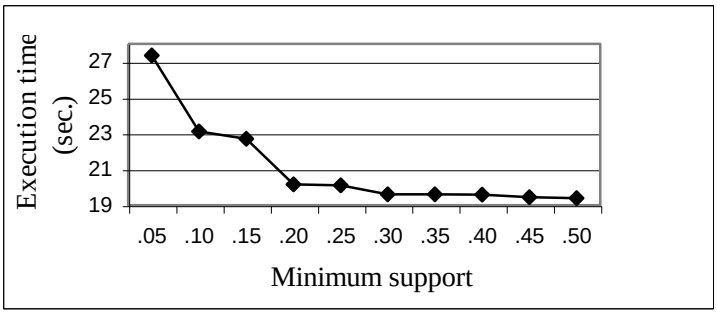

Figure 5.8 Execution time versus minimum support (*retail*)

By comparing Figures 5.1-5.4, one notes that the steepness of a graph increases as the size of branch databases increase. Similar observation holds true for Figures 5.5-5.8.

In Section 5.3.1 we have explained the concept of interesting overall influence. Given a threshold value of γ, we have counted the number of overall influences. In Figures 5.9-5.12 we have shown how the number of interesting overall influence decreases over the increase of the minimum influence level.

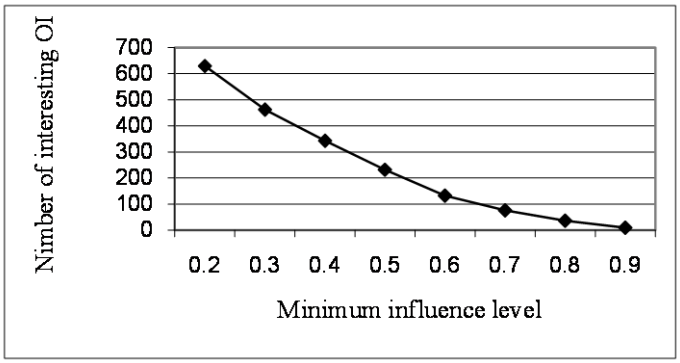

Figure 5.9 Number of interesting *OI* values versus γ at *supp* = 0.2 (*mushroom*)

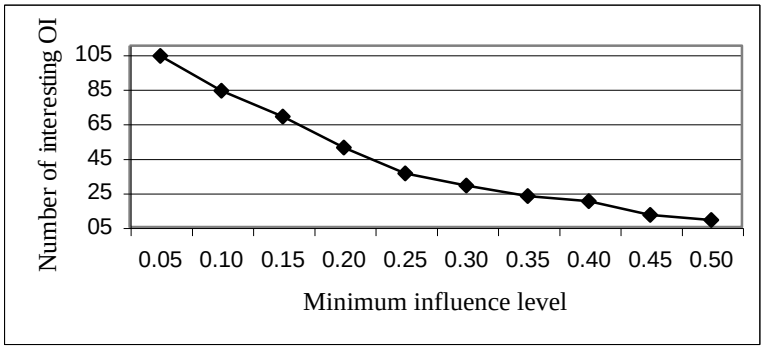

Figure 5.10 Number of interesting *OI* values versus γ at *supp* = 0.12 (*ecoli*)

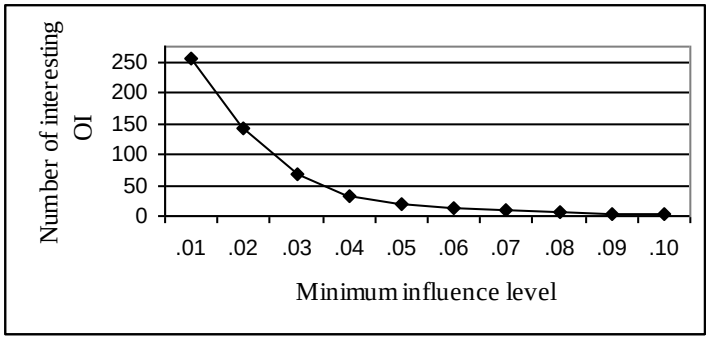

Figure 5.11 Number of interesting *OI* values versus γ at *supp* = 0.015 (*random-68*)

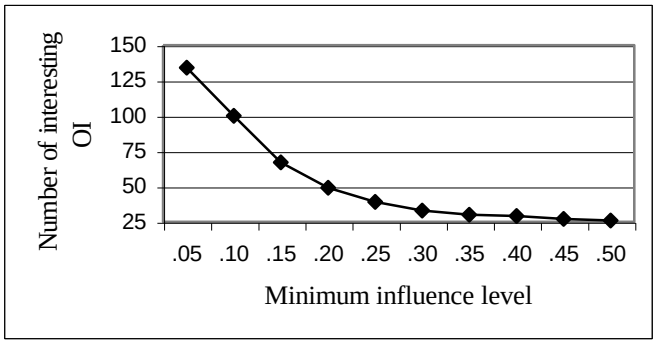

Figure 5.12 Number of interesting *OI* values versus γ at *supp* = 0.02 (*retail*)

Figures 5.9-5.12 also provide another type of insight. As the size of a transaction increases, the number of interesting overall influences also increase, provided the number of transactions in a branch database and the level of overall influence remain constant. The average transaction length of mushroom branch databases is significantly higher than that of other branch databases. The mining algorithm generates a large number of interesting overall influences even at minimum influence level 0.2.

We have taken specific items in different databases in Table 5.12. Based on the requirement of association analysis one could choose specific items in time databases.

Table 5.12 Specific items in different databases

M	*E*	*R*	*Rt*
SI = {1,2,3,6,9,10, 11,13,16, 23}	*SI* = {37,39,40,41,42, 44,48,49,50,51}	*SI* = {1,2,3,4,5, 6,7,8,9,10}	*SI* = {0,1,2,3,4,5, 6,7,8,9}

The influences of different items on a set of specific items in different databases are presented in Table 5.13. In the *mushroom* database, item 86 is the most influential item because 3 specific items are influenced by it. Item 48 in *ecoli* database exhibits a significant influence on the set of specific items. It shows that item 48 has high influence on 5 out of 10 specific items. In the same way one could conclude that item 18 in *random-68* is the most influential item with respect to the given set of specific items. 5 out of 10 specific items are influenced significantly by item 18. Item 413 influences 8 out of 10 specific items significantly in *retail* database. Therefore, it is the most influential item in *retail*.

Table 5.13 Influences of different items on a set of specific items in different databases

M (supp = 0.2)		*E* (supp = 0.12)		*R* (supp = 0.015)		*Rt* (supp = 0.03)	
γ	$x(\eta)$	γ	$x(\eta)$	γ	$x(\eta)$	γ	$x(\eta)$
0.3	86(3),34(3),36(3),39(2),59(2),63(2),2(2),93	0.07	48(5),37(2),50(1),42(1),44(1),39(1),	0.05	18(5),15(3),65(2),55(2),61(2),7(1),	0.05	413(8), 310(2),0(1), 1(1), 8(1),

5.8 Conclusion

The concept of positive influence might not be sufficient in many data analyses. One could perform an effective data analysis by using the measure of overall influence. Measuring influence over time becomes an important issue, since many companies possess data for a long period of time so that they could be exploited in an efficient manner. In this chapter, we have designed two algorithms using the measure of overall influence. The first algorithm reports all the significant influences in a database. In the second algorithm we have sorted items based on their influences on a set of specific items. Such analyses might be interesting since the proposed measure of influence considers both positive and negative influence of an itemset on another itemset.

Chapter 6

Conclusion

In this thesis we have analysed different time-stamped datasets. Temporal aggregation and database partitioning are two important data processing tasks applied to various problems from time to time. A database grown over a long period of time could be viewed as a collection of time databases. Temporal aggregation requires accumulation of values of various attributes in these time databases. A database grown over a long period of time could be large. Sometimes we need to divide a large database into smaller databases to carry out a data analysis, or to recognize relevant patterns. For example, a large temporal database could be divided into yearly databases, since a season re-appear on a yearly basis. Afterwards, one could make an analysis whether the yearly patterns are partial or fully periodic. Thus, mining a large temporal database could be considered as a problem of mining multiple time databases. Therefore, database partitioning is an important preprocessing task.

Over last two decades researchers have proposed many interesting temporal patterns such as frequent pattern, temporal association rule, event, sequential pattern, episode, and temporal relational interval pattern. Temporal patterns invented so far might not be an exhaustive list of temporal patterns, since temporal data generation is an ongoing process in different domains. In this context, we have also proposed few patterns mentioned in the following paragraphs. Also, various temporal data mining tasks such as prediction, clustering, classification, search and retrieval, and pattern discovery have been applied to various problems. In this thesis we have applied data mining tasks viz., clustering, pattern discovery, and other association analyses.

The variation of sales of an item over time is important information. Many decisions could be influenced by support information of items whose variations are less. For example, company could mine the association rules where antecedent of the rule is stable item. Then it could launch

"Buy a product, get a gift" sales promotion campaigns, where a gift is a stable item and the basic product has high margin rate. Sometime a set of products are defined and sold together with a discount where one product is stable item and the rest of the products have high margin rate.

For the purpose of finding the support variation of items, we divided a database into several yearly databases. In this connection a model of mining global patterns in multiple time-stamped databases has been proposed. Using yearly supports of an item, we have proposed the notion of stability of an item. The degree of stability is based on the variations of means and autocovariances. The items whose degree of stability is less than the user-defined threshold are called as stable items. Stable items are useful for modeling various strategies of an organization. Clustering relevant objects is an important task of many decision support systems. We have designed an algorithm for clustering items in multiple databases based on degree of stability. Also we have proposed the notion of best cluster by considering average degree of variation of a class. The experimental results show that the proposed clustering technique is effective and promising.

Recognition of patterns in temporal database is an important task. Over the years, there may exist many ups and downs in sales of an item. We observe that the change in sales series of an item at a particular year could be increasing, decreasing and altering. A new type of pattern, called notch, has been proposed based on the variation of sales of an item over the years in a time-stamped database. Based on this pattern, we have proposed generalized notch, and a special generalized notch, called iceberg, in sales series of an item. When the height and width of a generalized notch exceeds user-defined threshold it is considered as an iceberg. Iceberg notch represents a special sales pattern of an item over time. Study of such patterns is important to

understand the purchase behaviour of customers. Also, it helps identifying the reason of such behaviour. We have designed an algorithm for mining interesting icebergs in a time-stamped database.

A calendar-based periodic pattern is dependent on a schema of a calendar. We assume that the schema of the calendar-based pattern is based on day, month and year. In Chapter 4, we have proposed several improvements on the existing algorithm for identifying calendar-based periodic patterns. We have proposed a hash-based data structure for storing and managing the periodic patterns. We have extended the notion of certainty factor by incorporating support information for effective analysis of overlapped intervals. In addition to the proposed modified algorithm, we have also designed an algorithm for finding periodicity of calendar-based patterns. We have presented an extensive analysis on three datasets. We also have analysed the constraints *mininterval*, *minsupp* and *maxga*p associated with each interval. We have provided a comparative analysis, and shown that our algorithm outperforms the existing algorithm.

Influence of items on some other items might not be the same as the association between these sets of items. Measuring influence over time becomes an important issue, since many organizations possess data over a long period of time. The concept of positive influence might not be sufficient in many data analyses. One could perform an effective data analysis by using the measure of overall influence. In Chapter 5 we have proposed a measure, called *OI*, for measuring overall influence between two itemsets in a database. The proposed measure is effective, since it considers both positive and negative influence between two itemsets. We have designed two algorithms for influence analysis involving specific items in a database. The first algorithm reports all the significant influences in a database. In the second algorithm, we have sorted items based on their influences on a set of specific items.

In summary, we have mined different time-stamped datasets, and provided various types of data analyses. Generation of time-dependent data seems to be a natural phenomenon, and hence the analysis of such data always remains an active area of research.

References

Adhikari, A., Rao, P. R., "Enhancing Quality of Knowledge Synthesized from Multi-Database Mining", *Pattern Recognition Letters*, 28(16), pp. 2312 - 2324, 2007a.

Adhikari, A., Rao, P. R., "Study of select items in multiple databases by grouping", *In: Proceedings of the International Conference on Artificial Intelligence*, pp. 1699 - 1718, 2007b.

Adhikari, A., Rao, P. R., "A framework for mining arbitrary Boolean expressions induced by frequent itemsets", *In: Proceedings of the International Conference on Artificial Intelligence*, pp. 5 - 23, 2007c.

Adhikari, A. Rao, P. R., "Synthesizing Heavy Association Rules from Different Real Data Sources", *Pattern Recognition Letters*, 29(1), pp. 59-71, 2008a.

Adhikari, A., Rao, P. R., "Efficient clustering of databases induced by local patterns", *Decision Support Systems*, 44 (4), pp. 925 - 943, 2008b.

Adhikari, A., Rao, P. R., "Mining conditional patterns in a database", *Pattern Recognition Letters*, 29(10), pp. 1515 - 1523, 2008c.

Adhikari, J., Rao, P. R., Adhikari, A., "Clustering items in different data sources induced by stability", *The International Arab Journal of Information Technology*, 6(4), pp. 394 - 402, 2009.

Adhikari, J., Rao, P. R., "Measuring influence of an item in a database over time", *Pattern Recognition Letters*, 31(1), pp. 179 - 187, 2010.

Adhikari, J., Rao, P. R., "Identifying calendar-based periodic patterns", *Emerging Paradigms in Machine Learning*, S. Ramanna, L. Jain and R. J. Howlett (editors), pp. 329–357, Springer, 2013.

Adhikari, J., Rao, P. R., Pedrycz, W., "Mining icebergs in time-stamped databases", *Indian International Conference on Artificial Intelligence*, pp. 639 - 658, 2011.

Agrawal R., Imielinski, T., Swami, A., "Mining association rules between sets of items in large databases", *In: Proceedings of ACM SIGMOD Conference Management of Data*, pp. 207 - 216, 1993.

Agrawal, R., Srikant, R., "Fast algorithms for mining association rules", *In: Proceedings of 20th Very Large databases (VLDB) Conference*, pp. 487 - 499, 1994.

Agrawal, R., Srikant, R., "Mining sequential patterns", *In: Proceedings of International Conference on Data Engineering*, pp. 3 - 14, 1995.

Aggarwal, C., Yu, P., "A new framework for itemset generation", *In: Proceedings of PODS*, pp. 18 - 24, 1998.

Ale, J. M., Rossi, G. H., "An approach to discovering temporal association rules", *In: Proceeding of ACM Symposium on Applied Computing*, pp. 294 - 300, 2000.

Allen, J. F., "Maintaining knowledge about temporal intervals", *Comm. of the ACM*, 26(11), pp. 832 - 843, 1983.

Antunes, C. M., Oliveira, A. L., "Temporal data mining: an overview", *Proceedings of the KDD'01 Workshop on Temporal Data Mining*, USA, pp. 1 - 13, 2001.

Aref, W. G., Elfeky, M. G., Elmagarmid, A. K., "Incremental, online, and merge mining of partial periodic patterns in time-series databases", *IEEE TKDE*, 16(3), pp. 332 - 342, 2004.

Atallah, M. J., Gwadera, R., Szpankowski, W., "Detection of significant sets of episodes in event sequences", *In: Proceedings 4th IEEE International Conference on Data Mining (ICDM)*, pp. 3 - 10, 2004.

Baruah, H. K., "Set superimposition and its application to the theory of fuzzy sets" *J. Assam Sciebce Soc*. 10(1 and 2), pp. 25 - 31, 1999.

Bettini, C., Wang, X. S., Jajodia, S., "A general framework for time granularity and its application to temporal reasoning", *Ann. Math. Artificial Intelligence*, 22(1-2), pp. 29 - 58, 1998.

Bettini, C., Jajodia, S., Wang, X. S., "Time granularities in databases, data mining, and temporal reasoning", *Springer*, 2000.

Bluman A.G., *Elementary Statistics*: *A Step by Step Approach*, Mcgraw Hill, 2006.

Böttcher, M., Hoppner, F., Spiliopoulou, "On Exploiting the Power of Time in Data Mining", *SIGKDD Explorations*, 10(2), pp. 3 - 11, 2008.

Box. G., Jenkins, G., Reinsel, G., *Time series analysis*, 3rd edition, Pearson Education, 2003.

Brin, S., Motwani, R., Ullman, J.D., Tsur, S., "Dynamic itemset counting and implication rules for market basket data", *In: Proceedings of the ACM SIGMOD International Conference on Management of Data*, pp. 255 - 264, 1997.

Bringmann, B., Zimmermann, A., "One in a million: picking the right patterns", Knowledge and Information Systems, *18(1), pp. 61 - 81, 2009.*

Brockwell, P. J., Davis, A. R., *Introduction to Time Series and Forecasting*, Springer, 2002.

Casas-Garriga, G., “Discovering unbounded episodes in sequential data”, *In: Proceeding of 7th Eur. Conf. on Principles and Practice of Knowledge Discovery in Databases (PKDD’03)*, pp 83 - 94, 2003.

Chen, W., Chundi, P., “Extracting hot spots of topics from time-stamped documents”, *Data and Knowledge Engineering*, pp. 642 - 660, 2011.

Cotofrei, P., Stoffel, K., “Time granularity in temporal data mining”, *Foundations of Computational Intelligence*, (6), pp. 67 - 96, 2009.

Date, C. J., Darwen, H., Lorentzos, N. A., Temporal data and the relational model, *Elsevier*, 1 - 422, 2002.

Dumas, M., Fauvet, M. C. Scholl, P. C., “Handling temporal grouping and pattern-matching queries in a temporal object model”, *CIKM*, pp. 424 - 431, 1998.

Estivill-Castro V., Yang J., “Fast and robust general purpose clustering algorithms”, *Data Mining and Knowledge Discovery*, 8(2), pp.127 - 150, 2004.

Euzenat, J., Montanari, A., Time Granularity, Chapter 3 of the Handbook of Temporal Reasoning in Artificial Intelligence, *Elsevier B. V.*, pp. 59 - 118, 2005.

Fink, E., Gandhi, H. S., “Important extrema of time series”, *SMC*, pp. 366 - 372, 2007.

Frequent itemset mining dataset repository. http://fimi.cs.helsinki.fi/data.

Garofalakis, M. N., Rastogi, R., Shim, K., “SPIRIT: Sequential pattern mining with regular expression constraints”, *In: Proceedings of VLDB*, pp. 223 - 234, 1999.

Gary, J. R., Petersen, A., “Analysis of cross category dependence in market basket selection”, *Journal of Retailing*, 76 (3), pp. 367 - 392, 2000.

Goralwalla, I. A., Leontiev, Y., Ozsu, M. T., Szafron, D., Combi, C., "Temporal granularity for unanchored temporal data", *CIKM*, pp. 414 - 423, 1998.

Goralwalla, Leontiev, Y., Ozsu, M. T., Szafron, D., and Combi, C., "Temporal granularity: completing the puzzle", *Journal of Intelligent Information System*, 16(1), pp. 41 - 63, 2001.

Greenwood, P. E., Nikulin, M. S., A Guide to Chi-Squared testing, first edition, *Wiley-Interscience*, 1996.

Han J., Pei J., Yiwen Y., "Mining frequent patterns without candidate generation", *In: Proceedings of ACM SIGMOD Conference Management of Data*, pp.1 - 12, 2000.

Han, J., Kamber, M., Data mining: Concepts and techniques, *Morgan Kauffmann*, 2006.

Han, J., Dong, G., Yin, Y., "Efficient mining on partial periodic patterns in time series database", *In: Proceedings of the ICDE*, pp. 106 - 115, 1999.

Hand, D. J., Kok, J. N., Berthold, M. R., "Advances in intelligent data analysis", *In: Proceedings of Third International Symposium*, IDA-99, Amsterdam, *Springer* 1999.

Hong, T. P., Wu, Y. Y., Wang, S. L., "An effective mining approach for up-to-date patterns", *Expert Systems with Applications*, (36), pp. 9747 - 9752, 2009.

Hsu, W., Lee, M. L., Wang, J., Temporal and Spatio-Temporal Data Mining, *IGI Publishing*, 2008.

Jain, A. K., Murty M. N., Flynn P. J., "Data clustering: A review", *ACM Computing Surveys*, 31(3), pp. 264 - 323, 1999.

KDD CUP 2000, http://www.ecn.purdue.edu / KDDCUP.

Kempe, S., Hipp, J., Lanquillon, C., Kruse, R., "Mining frequent temporal patterns in interval sequences", *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 16(5), pp. 645 - 661, 2008.

Keogh, E., "A fast and robust method for pattern matching in time series databases", *In: Proceedings of 9th International Conference on tools with AI* (*ICTAI*), pp. 578 -584,1997.

Keogh, E., Lonardi, S., Chiu, B., "Finding surprising patterns in a time series database in linear time and space", *SIGKDD*, pp. 23 - 26, 2002.

Keogh, E., Lin, J., Lee, S. H., Herle, H. V., "Finding the most unusual time series subsequence: algorithms and applications", *Knowledge and Information Systems* 11(1), pp.1 - 27, 2006.

Khan, M. S., Coenen, F., Reid, D., Patel, R., Archer, L., "A sliding windows based dual support framework for discovering emerging trends from temporal data", *Knowledge based System*, 23(4), pp. 316 - 322, 2010.

Knuth, D.E., The Art of computer Programming, sorting and searching, second edition, (3), *Addison-Wesley Professional*, 1998.

Lattner, A. D., Temporal Pattern Mining in Dynamic Environments, *IOS Press*, 2007.

Laxman, S., Sastry, P. S., "A survey of temporal data mining", 31(2), pp. 173 - 198, 2006.

Lee, C. H., Lin, C. R., Chen, M. S., "Sliding-window filtering: An efficient algorithm for incremental mining", *Proceedings of 10th International Conference on Information and Knowledge Management*, pp. 263 - 270, 2001.

Lee, G., Yang, W., Lee, J. -M., "A parallel algorithm for mining multiple partial periodic patterns", *Information Science*, 176(24), pp. 3591 - 3609, 2006.

Lee, Y. J., Lee, J. W., Chai, D., Hwang, B., Ryu, K. H., "Mining temporal interval relational rules from temporal data", *Journal of Systems and Software*, 82(1), pp. 155-167, 2009.

Leonard M., Wolfe B., "Mining transactional and time series data", *SUGI 30 Proceedings*, pp. 080 - 30, 2005.

Li, Y., Ning, P., Wang, X. S., Jajodia, S., "Discovering calendar-based temporal association rules", *Data and Knowledge Engineering*, 44(2), pp. 193 - 218, 2003.

Li, D., Deogun, J. S., "Discovering partial periodic sequential association rules with time lag in multiple sequences for predicton", LNCS, (3488), pp. 332 - 341, 2005.

Liu C. L., Elements of discrete mathematics, *McGraw-Hill*, 1985.

Liu B., Ma Y., Lee R., "Analyzing the interestingness of association rules from the temporal dimension", *IEEE International Conference on Data Mining*, pp. 377 - 384, 2001.

Mahanta, A. K., Mazarbhuiya, F. A., Baruah, H. K., "Finding locally and periodically frequent sets and periodic association rules", *In: Proceedings 1st International Conference on Pattern Recognition and Machine Intelligence*, LNCS, (3776), pp. 576 - 582, 2005.

Mahanta, A. K., Mazarbhuiya, F. A., Baruah, H. K., "Finding calendar-based periodic patterns", *Pattern Recognition Letters*, 29(9), pp. 1274 - 1284, 2008.

Manilla, H., Toivonen, H., Verkamo, L., "Discovery of frequent episodes in event sequences", *Data Mining Knowledge Discovery, International Journal*, 1(3), pp. 259-289, 1997.

Mitsa, T., Temporal Data Mining, *CRC Press*, 2010.

Moon, B., Lopez, I. F. V., Immanuel, V., "Efficient algorithms for large-scale temporal aggregation", *IEEE Transaction Knowledge Data Engineering*, 15(3), pp. 744 - 59, 2003.

Ozden, B., Ramaswamy, S., Silberschatz, A., "Cyclic association rules", *In: Proceedings of 14th International Conference on Data Engineering*, pp. 412- 421, 1998.

Pratt, K., Fink, E., "Search for patterns in compressed time series", *International Journal of Image and Graphics*, 2(1), pp. 89-106, 2002.

Roddick, J. F., Spillopoulou, M., "A Bibliography of temporal, spatial and spatio-temporal data mining Research", *ACM SIGKDD*, 1(1), pp. 34 - 38, 1999.

Roddick, J. F., Spiliopoulou, M., "A survey of temporal knowledge discovery paradigms and methods", *IEEE TKDE*, pp. 750 - 767, 2002.

Savasere, A., Omiecinski, E., Navathe, S., "An efficient algorithm for mining association rules in large databases", *In: Proceedings of the International Conference on Very Large Data Bases*, pp. 432 - 443, 1995.

Shapiro, P., "Discovery, analysis, and presentation of strong rules", *Knowledge Discovery in Databases*, pp. 229 - 248, 1991.

Singh, S., Stuart. E., "A pattern matching tool for time series forecasting", *In: Proceedings of 14th International Conference on Pattern Recognition*, Brisbane, pp. 103 - 105, 1998.

Smyth, P., Goodman, M., "An Information theoretic approach to rules induction from databases", *IEEE, TKDE*, 4(4), pp. 301 - 316, 1992.

Srikant, R., Agrawal, R., "Mining sequential patterns: generalizations and performance improvements", *EDBT*, pp. 3 - 17, 1996.

Tan P. N., Kumar V., Srivastava J., "Selecting the right interestingness measure for association patterns", *In: Proceedings of SIGKDD Conference*, pp. 32 - 41, 2002.

Tanbeer, S. K., Ahmed, C. F., Jeong, B. S., Lee, Y. K., "Discovering periodic-frequent patterns in transactional databases", *Pacific Asia Knowledge Discovery in Databases*, pp. 242 - 253, 2009.

Tansel, A. U., Clifford, J., Gadia, S. K., Jajodia, S., Segev, A., Snodgrass, R. T., Temporal Databases: Theory, Design, and Implementation, *Benjamin / Cummings*, 1993.

Terenziani, P., Snodgrass, R. T., "Reconciling point based and interval based semantics in temporal relational databases, A treatment of the telic/atelic distinction", *IEEE Transactions on knowledge and Data Engineering*, 16(5), pp. 540 - 551, 2004.

Toman, D., "Point vs. interval-based query languages for temporal databases", *In: Proceedings of the 15th ACM SIGACTSIGMOD-SIGART Symposium on Principles of Database Systems* (PODS), pp. 58 - 67, 1996.

Tsay, R. S., "Identifying multivariate time series models", *Journal of Time Series and Analysis,* (10), pp. 357 - 372, 1989.

UCI ML repository content summary. http://www.ics.uci.edu/~mlearn/MLSummary.html

Verma, K., Vyas, O. P., Vyas, R., "Temporal approach to association rule mining using T-tree and P-tree", LNCS, (3587), pp. 651 - 659, 2005

Wang, J., Han, J., "BIDE: Efficient mining of frequent closed sequences", *In ICDE'04 Proeedings of the 20th International Conference on Data Engineering*, pp. 79 - 90, 2004.

Wang, K., Zhang, J., Shen, F., Shi, L., "Adaptive learning of dynamic Bayesian networks with changing structures by detecting geometric structures of time series", *Knowledge and Information Systems*, 17(1), pp. 121 - 133, 2008.

Wu, S., Manber, U., "Fast Text Searching Allowing Errors, *Communications of the ACM*), 35(10), pp. 83 – 91, 1992.

Wu, X., Zhang S., "Synthesizing high-frequency rules from different data sources", *IEEE Transactions on Knowledge and Data Engineering*, 14(2), pp. 353 -367, 2003.

Wu, X., Zhang, C., Zhang, S., "Efficient mining of both positive and negative association rules", *ACM Transactions on Information Systems*, 22(3), pp.381 - 405, 2004.

Wu, X., Zhang, C., Zhang, S., "Database classification for multi-database mining", *Information Systems*, 30(1), pp. 71 - 88, 2005.

Yang, K., Shahabi, C., "On the stationarity of multivariate time series for correlation-based data", *In: Proceedings of ICDM*, pp. 805 - 808, 2005.

Zaki, M. J., "Sequence mining in categorical domains: Incorporating constraints", *CIKM*, pp. 422 - 429, 2000a.

Zaki, M. J., "Scalable algorithms for association mining", *IEEE Transactions on Knowledge and Data Engineering*, 12(3), pp. 372 - 390, 2000b.

Zaki, M. J., "SPADE: An efficient algorithm for mining frequent sequences", *Machine Learning Journal*, 42(1/2), pp. 31 - 60, 2001.

Zhang, T., Ramakrishnan, R., Livny, M., "BIRCH: A new data clustering algorithm and its applications", *Data Mining and Knowledge Discovery*, 1(2), pp.141 -182, 1997.

Zhang, S., Wu, X., Zhang, C., "Multi-database mining", *IEEE Computational Intelligence Bulletin*, 2(1), pp. 5 - 13, 2003.

Zhang, S., Zhang, C., Wu, X., Knowledge discovery in multiple databases *Springer*, 2004.

Zhang, S., Zhang, J., Zhang, C., "EDUA: An efficient algorithm in dynamic database mining", *Information Sciences*, 177(13), pp. 2756 - 2767, 2007.

Zimbrao, G., Moreira de Souza, J., Teixeira de Almeida, V., Arauja de Silva, W., An algorithm to discover calendar-based temporal association rules with item's lifespan restriction, *In: Proceedings of 8th ACM SIGKDD*, 2002.