Chapter 13
Identifying Calendar-Based Periodic Patterns

Jhimli Adhikari and P.R. Rao

Abstract. A large class of problems deals with temporal data. Identifying temporal patterns in these datasets is a natural as well as an important task. 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. In this paper, we have extended the concept of certainty factor by incorporating support information for effective analysis of overlapped intervals. We have proposed a number of improvements in the algorithm for identifying calendar-based periodic patterns. In this direction we have proposed a hash based data structure for storing and managing patterns. Based on our modified algorithm, we identify full as well as partial periodic calendar-based patterns. We provide a detailed data analysis incorporating various parameters of the algorithm and make a comparative analysis with the existing algorithm, and show the effectiveness of our algorithm. Experimental results are provided on both real and synthetic databases.

Keywords: Calendar-based pattern, Certainty factor, Overlapped interval, Periodic pattern, Temporal pattern, Time-stamped database.

13.1 Introduction

A large amount of data being collected every day has a temporal connotation. For example, databases those originate from transactions in a supermarket, logs in a network, transactions in a bank, and events related to manufacturing industry are

Jhimli Adhikari
Department of Computer Science, Narayan Zantye College,
Bicholim, Goa - 403 529, India
e-mail: jhimli_adhikari@yahoo.co.in

P.R. Rao
Department of Computer Science and Technology, Goa University, Goa - 403 206, India
e-mail: pralhadrrao@gmail.com

S. Ramanna, L.C. Jain, & R.J. Howlett (Eds.): Emerging Paradigms in ML, SIST 13, pp. 329–357.
springerlink.com

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 [5], periodical association rule [17], calendar association rule [18], calendar-based periodic pattern [20], and up-to-date pattern [12] 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 [22]. A supermarket manager may discover that turkey and pumpkin pie are frequently sold 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 occur. 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 a 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. [20] used set superimposition [8] 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, the elements of $(A \cap B)^{(2)}$ 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 patterns. While applying this concept authors have assumed equi-fuzzy intervals, 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 equi-fuzzy membership value 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 for the pattern. The *support* [3] of a pattern represents a fraction of transactions containing the pattern. A pattern is *frequent* if its support is greater than equal to a user-defined threshold, *minsupp*. The certainty factor and support of a pattern in an interval are two different concepts. For an effective analysis of overlapped regions, these two concepts need to be introduced along with an overlapped region. Thus, in this paper we propose an extended analysis of superimposed intervals. The main weak point of the aforementioned paper is that the concept of set superimposition is not necessary in the proposed algorithm. Therefore, we have proposed a modified algorithm for identifying full as well as 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 give 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 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 paper is organized as follows. We discuss related work in Section 2. In Section 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. Experimental results are provided in Section 5. We conclude the paper in Section 6.

13.2 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 paper we have dealt with calendar dates over the years.

Verma et al. [23] 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., item-id, support count and a hyper-link. In order to deal with the patterns in time-stamped databases we have proposed a hash-based data structure where at the first index level we store distinct years that appear in the transactions. Then we keep an array of pointers corresponding to every year in the index table. The *k-th* pointer of this array points to tables containing interesting itemsets of size *k*.

Lee et al. [15] 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. [18] 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. Our work differs from it, since we identify frequent itemsets along with the associated intervals. Then we use match ratio to determine whether a pattern is full periodic or partial periodic. Subsequently, Zimbrao et al. [25] reported a similar work. Authors incorporate multiple granularities of time intervals from which both cyclic and user-defined calendar patterns can be achieved. Ale and Rossi [6] proposed an algorithm to discover temporal association rules. In this algorithm, support of an item is calculated only during its lifespan. In the proposed work we compute and store supports of itemsets when they satisfy the requirements of the user.

Lee et al. [16] 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 the most two times. Our approach extracts both partial and full periodic patterns together by scanning the database repeatedly to find the higher-level patterns as done using apriori algorithm [4].

In the context of support definition, Kempe et al. [13] 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. [14] have proposed a new temporal data mining technique that can extract temporal interval relation rules from temporal interval data by using Allen's theory [7]. Authors designed a preprocessing algorithm for generalization of temporal interval data. Also, authors have proposed an algorithm for discovering a temporal interval relation. Although there are thirteen different types of relations between two intervals, in our work we have focused on only overlapped intervals to find locally frequent itemsets of larger size and detect periodicity of patterns.

Ozden et al. [21] proposed a method of finding patterns having periodic nature where the period has to be specified by the user. Han et al. [11] 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.

13.3 Calendar-Based Periodic Patterns

In Sections 1, 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 calendar-based pattern is based on day, month and year. 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. Thus it 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 a 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. Here 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 [9] 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 [9] in the database. Partial periodicity is a looser form of periodicity than full periodicity, and it also occurs more commonly in a real world database. A pattern is associated with a real number m ($0 < m < 1$), called match ratio [18] that reveals that 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. [20] have proposed match ratio in somewhat a different way. Authors have proposed match ratio as the number of intervals is divided by the number of years in the lifespan of the pattern for the purpose of mining yearly 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 [18].

We have discussed the concept of certainty factor in Section 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.

13.3.1 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. [20] 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 equi-fuzzy membership value. For example, if n intervals are superimposed then every interval has $1/n$ equi-fuzzy membership value and in an overlapped area the membership value will be added. The certainty of the pattern in the overlapped interval is more than the certainty in the other intervals.

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, while 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 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 lifespan 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 $[t_1, t_2]$.

Let a 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:

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

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 $r = 2, 3, \ldots, n$. Otherwise, the certainty value 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 $r = 3, 4, \ldots, n$.

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. The purpose of this example is to explain the proposed concept of extended certainty factor stated above. Let the years 1980, 1981, 1982 and 1983 be of our interest. We would like to check whether the pattern X is yearly periodic. Assume that the mining algorithm has reported X as 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 1980, 1981, 1982 and 1983, 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 1.

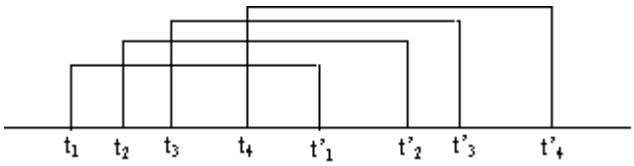

Fig. 1 Overlapped intervals for finding yearly pattern X

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 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 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 in a time-interval. Thus, for an effective analysis of a superimposed interval both the certainty factor and support information are needed in association with an interval.

13.3.2 Extending Certainty Factor with Respect to Other Intervals

In Figure 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 [20], 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 also becomes 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.

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 2.

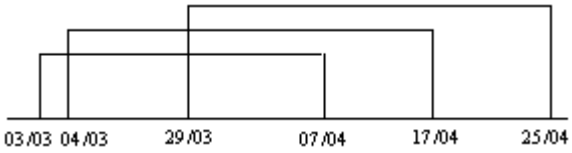

Fig. 2 Overlapped intervals for finding yearly pattern $\{a, c\}$

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 the certainty factor and *s-info* for a pattern. In Table 3 we present an analysis of intervals for finding yearly pattern $\{a, c\}$. •

Table 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.

13.4 Mining Calendar-Based Periodic Patterns

Itemsets in transactions could be considered as a basic type of pattern in a database. Many interesting patterns such as association rules [3], negative association rules [24], Boolean expressions induced by itemset [1] and conditional patterns [2] are based on itemset patterns. Some itemsets are 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 associated time-slots.

13.4.1 Improving Mining Calendar-Based Periodic Patterns

The goal of this paper is to study the existing algorithm, and to propose an effective algorithm by improving the limitations of the existing algorithm for mining calendar-based periodic patterns. As noted 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. Mahanta et al. [19] have proposed an algorithm for finding all the locally frequent itemsets of size one. While studying the algorithm we have found that some variables contradict their definitions. 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 the algorithm, seems to be not appropriate. 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 inappropriate. Moreover, the validity of the experimental results is low, since it is based on only one dataset. In view of improving the algorithm further we propose a number of modifications mentioned as follows: (i) The proposed algorithm makes corrections on the existing algorithm using the points noted 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.

13.4.2 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. We describe the data structure using Example 5 given below.

Example 5. Consider the database D of Example 1. Transactions consisting of items a, b, c, d, e, f, g, h, and i occurred in the years from 1990 to 1994. 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 3. 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 intervals. All the five years are stored in an index table at level 0. After level 0, we keep an array of pointers for every year. The first pointer corresponding to year 1990 points to a table containing interesting itemsets of size one, their intervals and local supports. The second pointer corresponding to year 1990, points to a table containing interesting itemsets of size two, their intervals and 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 3.

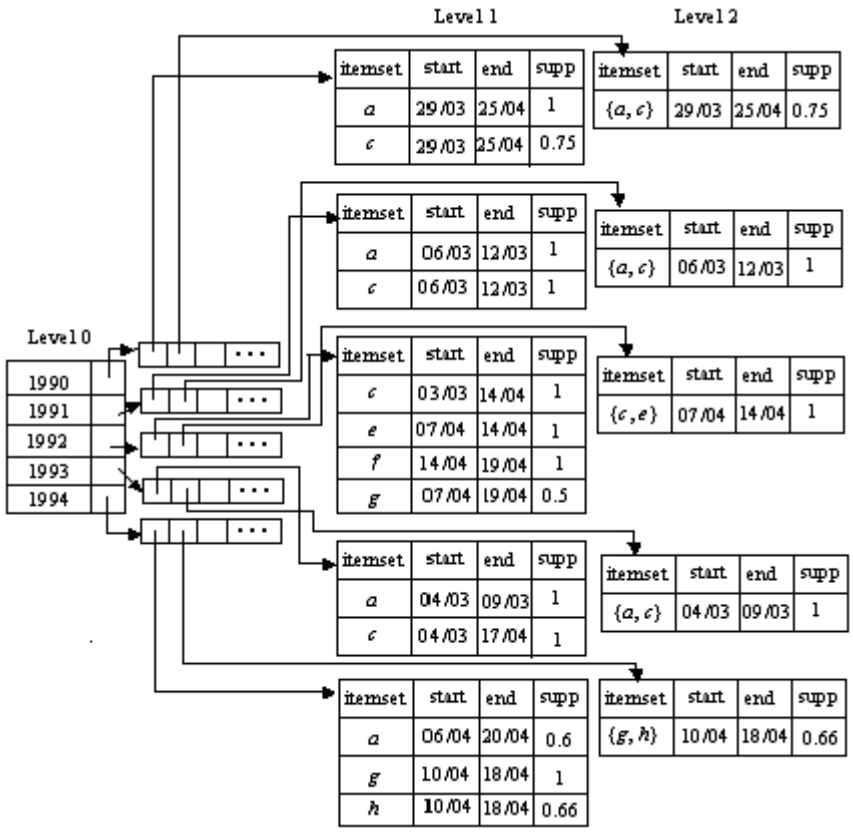

Fig. 3 Data structure used in the proposed algorithms

13.4.3 A Modified Algorithm

As mentioned in Section 4.1, we have proposed a number of improvements to the algorithm proposed by Mahanta et al. [19] 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 constraints *mininterval* and *minsupp* are retained. The modification made seems to be significant from the overall viewpoint 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 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 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, … . Variables *row* and *row_p* are used to index arrays *itemset_addr* and *level_p* respectively, *p* = 0, 1, 2, … . 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 Algorithm 1 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* (*D*, *maxgap*, *mininterval, minsupp*)
Inputs: *D, maxgap, mininterval, minsupp*
D: database to be mined
minsupp: as defined in Section 1
maxgap, mininterval: as defined in Section 3.2
Outputs:
Locally frequent items, their intervals and supports as mentioned in Figure 3
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:      store the k-th item, its firstseen, lastseen and local support at level_1[row_1];
14:       increase row_1 by 1;
15:       end if {12}
16:   end for {11}
17:   row_1 = 1;
18:   increase row_0 by 1; increase row by 1;
19:   level_0[row_0][1] = year(t.date); level_0[row_0][2] = itemset_addr[row][1];
20:   for k = 1 to n do
21:     lastseen[k] = 0; itemIntervalFreq[k] = 0; nTransInterval[k] = 0;
22:     end for
23:   end if {10}
24:   for k = 1 to n do
25:     if (item[k] ∈ t) then
26:       increase nItemsTrans by 1;
27:       if (lastseen[k] = 0) then
28:         initialize both lastseen[k] and firstseen[k] by t.date;
          initialize both itemIntervalFreq[k] and nTransInterval[k] by 1;
29:       else if ( | t.date − lastseen[k] | ≤ maxgap ) then
30:             lastseen[k] = t.date;
31:             increase itemIntervalFreq[k] by 1; increase nTransInterval[k] by 1;
32:             end if
33:       else if ( | lastseen[k] − firstseen[k] | ≥ mininterval) and
    (itemIntervalFreq[k] / nTransInterval[k] ≥ minsupp) then
34:   store the k-th item, its firstseen, lastseen and local support at level_1[row_1];
35:           increase row_1 by 1;
36:           initialize both lastseen[k] and firstseen[k] by t.date;
37:           initialize both itemIntervalFreq[k] and nTransInterval[k] by 0;
38:           end if {33}
39:       end if {27}
40:     else increase nTransInterval[k] by 1;
41:     end if {25}
42:    if (nItemsTrans = transLength) then exit from for-loop; end if
43:   end for {24}
44:   read a transaction t ∈ D;
45: end while {08}
46: for k = 1 to n do
```

47: **if** (|*lastseen*[*k*] − *firstseen*[*k*]| ≥ *mininterval*) **and**
(*itemIntervalFreq*[*k*] / *nTransInterval*[*k*] ≥ *minsupp*) **then**
48: store the *k*-th *item*, its *firstseen*, *lastseen* and local support at *level_1*[*row_1*];
49: increase *row_1* by 1;
50: **end if** {47}
51: **end for** {46}
52: 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 8-45 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 11-16. We retain those intervals that satisfy criteria of *mininterval* and *minsupp*. Lines 17-22 assign the necessary initializations for a different year. Lines 25-41 are repeated for each item in the current transaction. Line 27 checks whether the item is first time seen in the transaction and the necessary assignment is done in line 28. Lines 29-32 determine whether the current transaction-date is coming under the current interval by comparing the difference between *t.date* and *lastseen* with *maxgap*. Lines 33-38 construct an interval and compute the local support. Line 42 avoids the unnecessary repetition by comparing the transaction length. Line numbers 46-51 close all the last intervals for last year. Line 52 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 25. Consider a database containing 10,000 items. Let the current transaction be of length 20 and these items are within the first 100 items. Then the *for-loop* at line number 24 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 [4]. We use array *level_1* to generate the candidate sets at the 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 number 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 such as *maxgap*, *mininterval* and *minsupp*.

Algorithm 2. Mine locally frequent itemsets at higher level and the associated 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, the associated intervals and supports as mentioned in Figure 3
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** $((l_1[1] = l_2[1]) \wedge \ldots \wedge (l_1[k-2] = l_2[k-2]) \wedge (l_1[k-1] < l_2[k-1]))$ **then**
07: $c = l_1 \bowtie l_2$; $C_k = C_k \bigcup c$;
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 3 completely. One can use *S* to determine whether an itemset pattern is fully / partially periodic.

13.5 Experimental Studies

We have carried out several experiments for mining calendar-based periodic patterns on different databases. All the experiments have been implemented on a 2.4 GHz, core i3 processor with 4 GB of memory, running Windows 7 HB, using Visual C++ (version 6.0) software. We present experimental results using *retail* [10], *BMS-WebView-1* [10], and *T10I4D100K* [10] databases. Since the records in these databases contain only items purchased in transactions, we have attached time-stamps randomly as calendar date for the transactions. The characteristics of the databases are given in Table 4.

Table 4 Database characteristics

D	*NT*	*ALT*	*AFI*	*NI*	Size in Megabytes
retail	88,162	11.31	60.54	16,470	3.97
BMS-WebView-1	1,49,639	2.00	44.57	6,714	1.97
T10I4D100K	1,00,000	11.10	1276.12	870	3.83

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 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 the associated periodicities. We also compute certainty factor and match ratio of a pattern with respect to overlapped intervals.

Table 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

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 6 and 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. Although, it is peculiar but it remains also an artificial phenomenon, since the time-stamps are enforced by us. There are many itemsets such as {16217} are frequent in many years with non-overlapping intervals. In Table 6, we present itemsets of size one that are also part of interesting itemsets of size two as shown in Table 7. 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 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 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 8 and 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 a smaller *minsupp*. From Table 9 one could observe that itemset

{33449, 33469} shows periodicity by appearing two times in six years and the remaining interesting itemsets are reported for a year only.

Table 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 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 10 we present yearly periodic itemsets of size one for *T10I4D100K* database. In this database 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 in 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 aspects.

Table 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

13.5.1 Selection of Mininterval and Maxgap

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.

13.5.1.1 Mininterval

In the following experiments we would like to analyse the effect of *mininterval* for given *maxgap* and *minsupp*. We observe in Figures 4, 5 and 6, 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 data source to another. In *retail* many itemsets are locally frequent for longer period of time. In Figure 4 we observe that there exists 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.

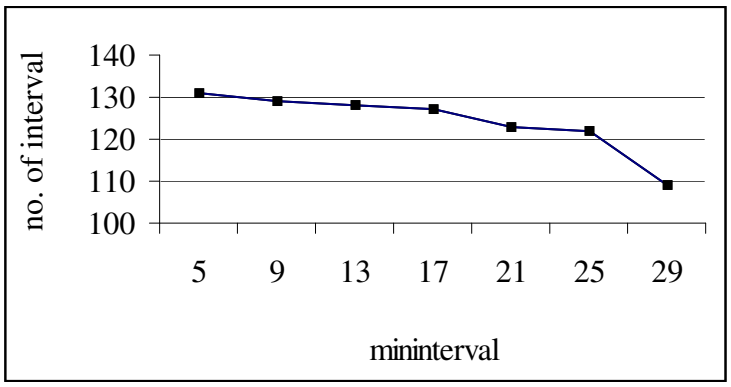

Fig. 4 *Retail* (*minsupp* = 0.25, *maxgap* = 7)

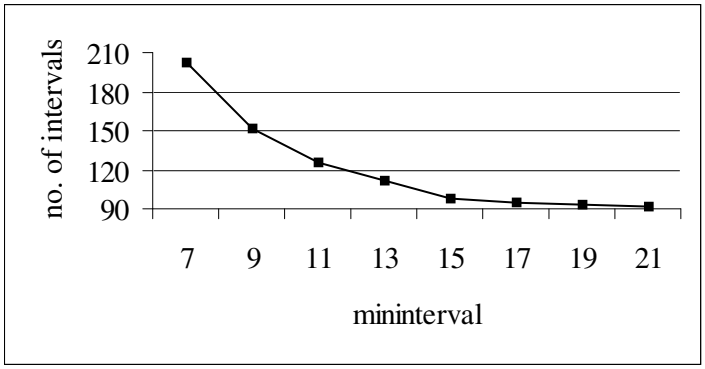

Fig. 5 *BMS-WebView-1* (*minsupp* = 0.06, *maxgap* = 7)

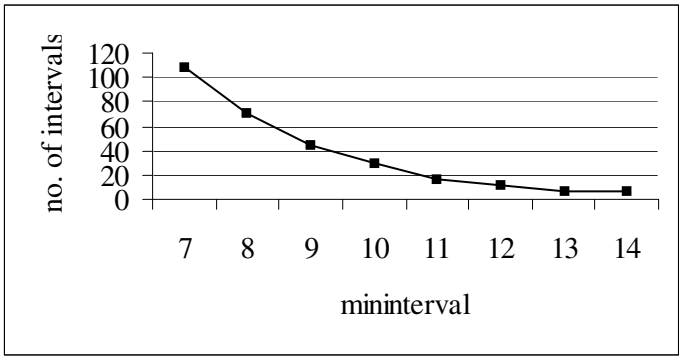

Fig. 6 *T10I4D100K* (*minsupp* = 0.13, *maxgap* = 7)

13.5.1.2 Maxgap

In view of analyzing *maxgap* parameter, we present graphs of the number of intervals versus *maxgap* at given *minsupp* and *mininterval* in Figures 7, 8, and 9. 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.

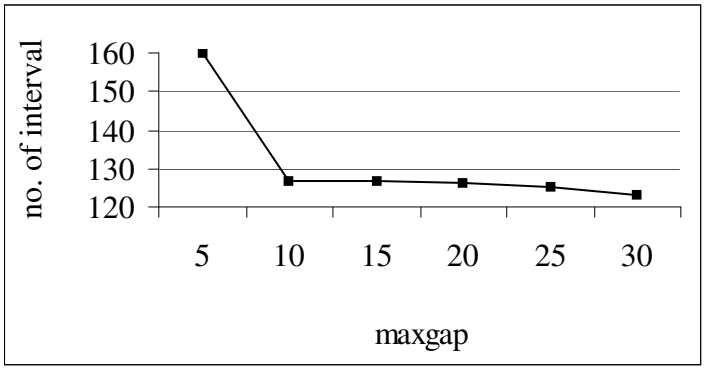

Fig. 7 *Retail* (*minsupp* = 0.25, *mininterval* = 10)

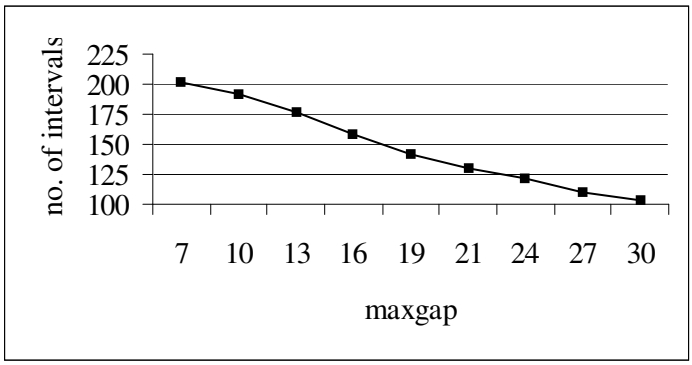

Fig. 8 *BMS-WebView-1*(*minsupp* = 0.06, *mininterval* = 7)

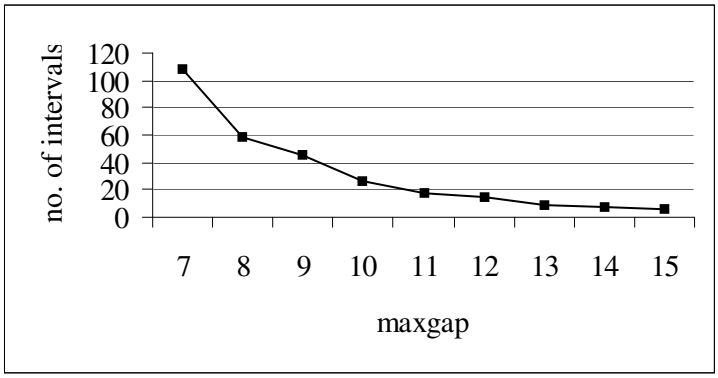

Fig. 9 *T10I4D100K* (*minsupp* = 0.13, *mininterval* = 7)

13.5.2 Selection of Minsupp

The number of intervals and minimum support are inversely related to given *maxgap* and *mininterval*. We observe this phenomenon in Figures 10, 11 and 12. When the value of *maxgap* 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.

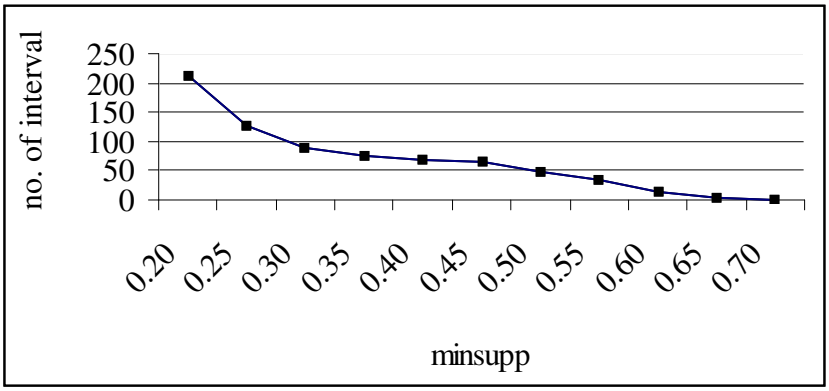

Fig. 10 *Retail* (*mininterval* = 10, *maxga*p = 12)

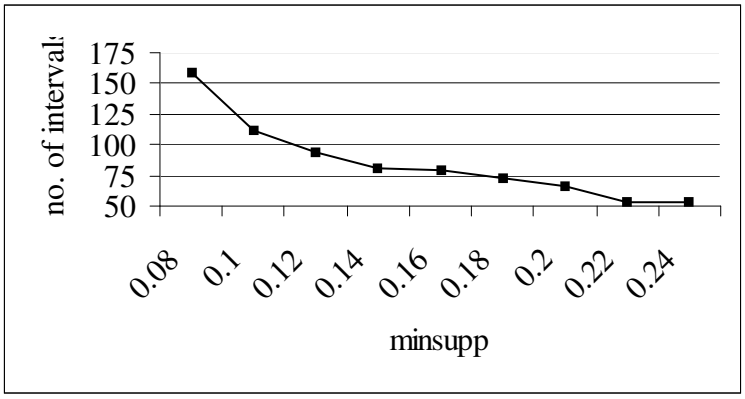

Fig. 11 *BMS-WebView-1*(*mininterval* = 7, *maxga*p = 10)

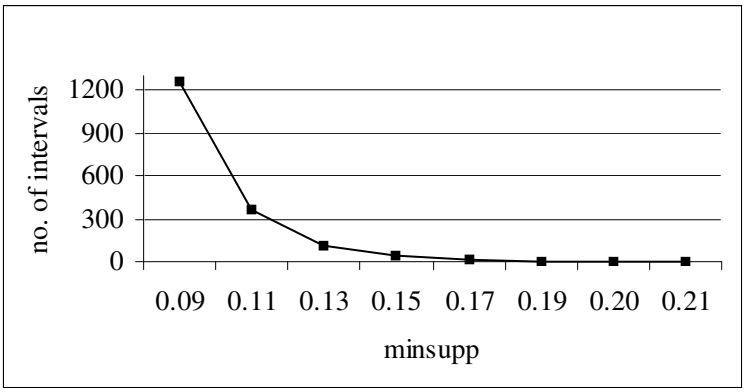

Fig. 12 *T10I4D100K* (*mininterval* = 7, *maxga*p = 9)

13.5.3 Performance Analysis

In this section, we present the performance of our algorithm and compare it with existing algorithm for mining calendar-based periodic patterns. To measure the performance, two experiments have been conducted. In the first experiment, we have measured the scalability of the two algorithms with respect to different database sizes. In the second experiment, we have measured the scalability of the two algorithms with respect to different support thresholds. In Figures 13, 14 and 15, we have shown the relationship between the database size and execution time for mining periodic patterns. We observed that the number of patterns increases as the number of transactions increases. Thus, the execution time increases with the increase of database size. Initially both the algorithms take almost equal amount of time. In Figures 13 and 14 we observe that execution time for mining 88162

transactions of *retail* and 1,49,639 transactions of *BMS-WebView-1* take nearly equal amount of time. The reason is that the average length of transactions in *retail* is more than that of *BMS-WebView-1*. Therefore, the execution time is not only dependent on the size of the database, but also depends on the factors such as *ALT* and *NI*. The experimental results in Figures 13, 14 and 15 show that our algorithm performs better than the existing algorithm.

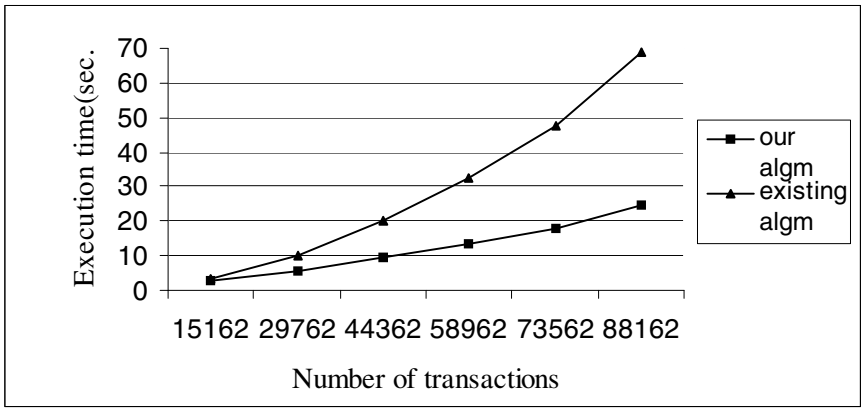

Fig. 13 Execution time vs. size of database at *minsupp* = 0.25, *mininterval* = 8, *maxgap* = 10 (*retail*)

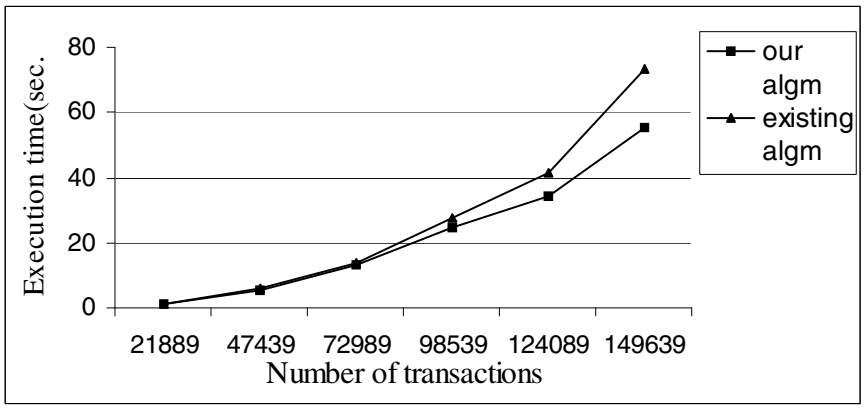

Fig. 14 Execution time vs. size of database at *minsupp* = 0.1, *mininterval* = 7, *maxgap* = 10 (*BMS-WebView-1*)

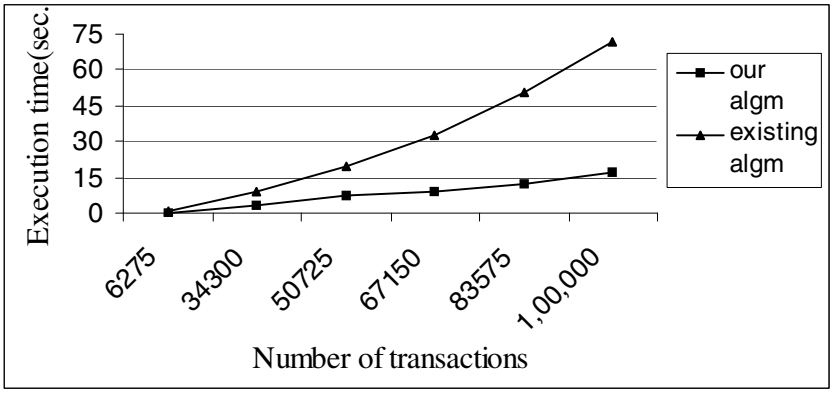

Fig. 15 Execution time vs. size of database at *minsupp* = 0.13, *mininterval* = 7, *maxga*p = 10 (*T10I4D100K*)

In Figures 16, 17, and 18, we have presented another comparison by considering *minsupp* threshold. When the minimum support increases the number of frequent itemsets decreases and so the execution time also decreases. The experimental results have shown that the execution time of both the algorithms decrease slowly when the support threshold increases, and our algorithm takes less time than the existing algorithm.

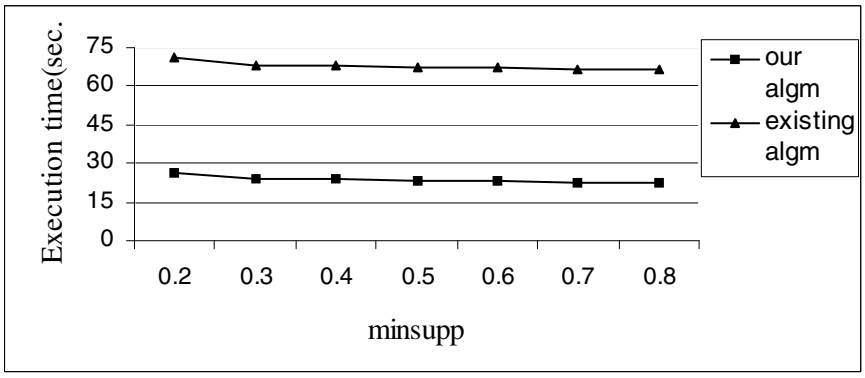

Fig. 16 Execution time vs. *minsupp* (*mininterval* = 8, *maxga*p = 10) for *retail*

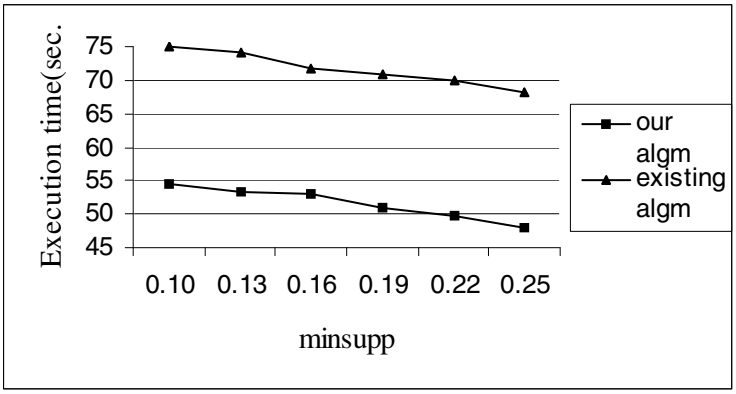

Fig. 17 Execution time vs. *minsuopp* (*mininterval* = 7, *maxga*p = 10) for *BMS-WebView-1*

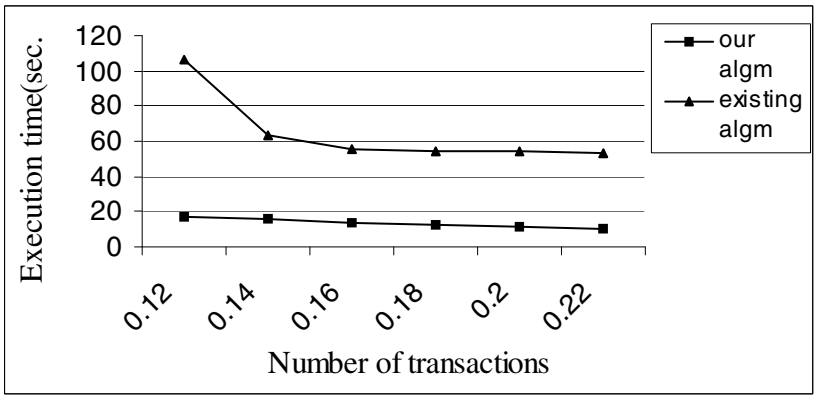

Fig. 18 Execution time vs. *minsupp* (*mininterval* = 7, *maxga*p = 10) for *T10I4D100K*

13.6 Conclusions

In this paper we have proposed modifications to the existing algorithm for mining locally frequent itemsets along with the set of intervals and associated supports. We have also extended the concept of certainty factor for a detailed analysis of overlapped intervals. We have proposed a number of improvements on the existing algorithm for finding calendar-based periodic patterns. For managing locally frequent itemsets effectively we have introduced a hash based data structure. We have presented an extensive data analysis by involving constraints such as *mininterval*, *minsupp* and *maxga*p. In addition we have compared our algorithm with the existing algorithm. Experimental results show that our algorithm executes faster than the existing algorithm. Experimental results also report that whether a periodic pattern is full or partial. The proposed algorithm can also be used to extract yearly, monthly, weekly and daily calendar-based patterns.

Acknowledgement. The first author would like to thank the University Grants Commission (UGC), Government of India, for having sponsored her on faculty improvement programme with leave to take up full time research.

References

1. 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 (2007)
2. Adhikari, A., Rao, P.R.: Mining conditional patterns in a database. Pattern Recognition Letters 29(10), 1515–1523 (2008)
3. Agrawal, R., Imielinski, T., Swami, A.: Mining association rules between sets of items in large databases. In: Proceedings of the ACM SIGMOD Conference Management of Data, pp. 207–216 (1993)
4. Agrawal, R., Srikant, R.: Fast algorithms for mining association rules. In: Proceedings of 20th Very Large databases (VLDB) Conference, pp. 487–499 (1994)
5. Agrawal, R., Srikant, R.: Mining Sequential Patterns. In: Proceedings of International Conference on Data Engineering (ICDE), pp. 3–14 (1995)
6. Ale, J.M., Rossi, G.H.: An approach to discovering temporal association rules. In: Proceedings of ACM Symposium on Applied Computing, pp. 294–300 (2000)
7. Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM 26(11), 832–843 (1983)
8. Baruah, H.K.: Set superimposition and its application to the theory of fuzzy sets. J. Assam Science Soc. 10(1 and 2), 25–31 (1999)
9. Bettini, C., Jajodia, S., Wang, S.X.: Time Granularities in Databases. In: Data Mining and Temporal Reasoning. Springer (2000)
10. Frequent itemset mining dataset repository, `http://fimi.cs.helsinki.fi/data`
11. Han, J., Dong, G., Yin, Y.: Efficient mining on partial periodic patterns in time series database. In: Proceedings of Fifteenth International Conference on Data Engineering, pp. 106–115 (1999)
12. Hong, T.P., Wu, Y.Y., Wang, S.L.: An effective mining approach for up-to-date patterns. Expert Systems with Applications (36), 9747–9752 (2009)
13. 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), 645–661 (2008)
14. Lee, J.W., Lee, Y.J., Kim, H.K., Hwang, B.H., Ryu, K.H.: Discovering temporal relation rules mining from interval data. In: Proceedings of the 1st Euro Asian Conference on Advance in Information and Communication Technology, Iran, (2002)
15. 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), 155–167 (2009)
16. Lee, G., Yang, W., Lee, J.M.: A parallel algorithm for mining multiple partial periodic patterns. Information Science 176(24), 3591–3609 (2006)
17. Li, D., Deogun, J.S.: Discovering Partial Periodic Sequential Association Rules with Time Lag in Multiple Sequences for Prediction. In: Hacid, M.-S., Murray, N.V., Raś, Z.W., Tsumoto, S. (eds.) ISMIS 2005. LNCS (LNAI), vol. 3488, pp. 332–341. Springer, Heidelberg (2005)

18. Li, Y., Ning, P., Wang, X.S., Jajodia, S.: Discovering calendar-based temporal association rules. Data and Knowledge Engineering 44(2), 193–218 (2003)
19. Mahanta, A.K., Mazarbhuiya, F.A., Baruah, H.K.: Finding Locally and Periodically Frequent Sets and Periodic Association Rules. In: Pal, S.K., Bandyopadhyay, S., Biswas, S. (eds.) PReMI 2005. LNCS, vol. 3776, pp. 576–582. Springer, Heidelberg (2005)
20. Mahanta, A.K., Mazarbhuiya, F.A., Baruah, H.K.: Finding calendar-based periodic patterns. Pattern Recognition Letters 29(9), 1274–1284 (2008)
21. Ozden, B., Ramaswamy, S., Silberschatz, A.: Cyclic association rules. In: Proceedings of 14th International Conference on Data Engineering, pp. 412–421 (1998)
22. Roddick, J.F., Spiliopoulou, M.: A survey of temporal knowledge discovery paradigms and methods. In: IEEE TKDE, pp. 750–767 (2002)
23. Verma, K., Vyas, O.P., Vyas, R.: Temporal Approach to Association Rule Mining Using T-Tree and P-Tree. In: Perner, P., Imiya, A. (eds.) MLDM 2005. LNCS (LNAI), vol. 3587, pp. 651–659. Springer, Heidelberg (2005)
24. Wu, X., Zhang, C., Zhang, S.: Efficient mining of both positive and negative association rules. ACM Transactions on Information Systems 22(3), 381–405 (2004)
25. Zimbrao, G., de Souza, J.M., de Almeida, V.T., de Silva, W.A.: An algorithm to discover calendar-based temporal association rules with item's lifespan restriction. In: Proceedings of 2nd Workshop on Temporal Data Mining (2002)