On the Discovery of Interesting Patterns in Association Rules

Sridhar Ramaswamy
Bell Labs
sridhar@bell-labs.com

Sameer Mahajan
Informix Corp.
sameer@informix.com

Avi Silberschatz
Bell Labs
avi@bell-labs.com

Abstract

Many decision support systems, which utilize association rules for discovering interesting patterns, require the discovery of association rules that vary over time. Such rules describe complicated temporal patterns such as events that occur on the "first working day of every month." In this paper, we study the problem of discovering how association rules vary over time. In particular, we introduce the idea of using a *calendar algebra* to describe complicated temporal phenomena of interest to the user. We then present algorithms for discovering *calendric association rules*, which are association rules that follow the patterns set forth in the user supplied calendar expressions. We devise various optimizations that speed up the discovery of calendric association rules. We show, through an extensive series of experiments, that these optimization techniques provide performance benefits ranging from 5% to 250% over a less sophisticated algorithm.

1 Introduction

Recent advances in data collection and storage technology have made it possible for many companies to keep vast amounts of data relating to their business online. At the same time, the availability of cheap computing power has also made some automatic analysis of this data feasible. This activity is commonly referred to as *data mining*.

One major application domain of data mining is in the analysis of transactional data. It is assumed that the database system keeps information about user transactions, where each transaction is a collection of data items (e.g., milk, break, eggs, etc.). In this setting, *association rules* capture inter-relationships between various data items. An association rule captures the notion of a set of data items occurring together in transactions. For example, in a database maintained by a supermarket, an association rule might be of the form:

"beer $\rightarrow$ chips (support: 3%, confidence: 87%),"

which means that 3% of all database transactions contain the data items beer and chips, and 87% of the transactions that have the item "beer" also have the item "chips" in them. The two percentage parameters above are commonly referred to as "support" and "confidence" respectively.

Typically, the data mining process is controlled by a user who sets minimum thresholds for the support and confidence parameters. The user might also impose other restrictions, like restricting the search space of items, in order to guide the data mining process.

Following the pioneering work of [AIS93], discovery of association rules has been extensively studied in [AS94, SA95, HF95, SON95, PCY95, SA96, FMMT96, Toi96]. However, all the above work treat all the data as one large segment, with no attention paid to segmenting the data over different time intervals. To illustrate, let us return to our previous example. It may be the case that beer and chips are sold together primarily between 6PM and 9PM on week days. Therefore, if we segment the data over the two intervals 7AM–6PM and 6PM–9PM and consider only the data from weekdays, we may find that the support for the beer and chips rule in the segment 6PM–9PM jumps to 50%.

From the above example we can conclude that although an association rule may have the user specified minimum confidence and support within the entire time spectrum, analysis of the data in finer time granularity may reveal that the association rule exists only in certain time intervals, and does not occur in the remaining time intervals. Even casual observation of many association rules over monthly data may disclose interesting patterns in their behavior over time. Detecting these patterns would reveal interesting information that in turn can be used for analysis, prediction and decision making.

In [ÖRS98], we examined this problem in the context of detecting when association rules occur periodically over time. In particular, we introduced the problem of mining for "cyclic association rules" and presented algorithms for efficiently detecting such rules. However, periodicity has limited power in describing real-life variations. On a day-to-day basis, humans deal with complicated patterns that cannot be described by simple periodicities. For example, a concept as simple as the first working day of every month cannot be described by cycles. Further, the model used

Proceedings of the 24th VLDB Conference
New York, USA, 1998

in [ÖRS98] was fairly rigid, failing to capture a rule that would exhibit a pattern "most" of the time but not all the time. Finally, the model did not deal with multiple units of time, like days, weeks, etc. that occur naturally.

In this paper, we generalize the work in [ÖRS98] in several important directions:

- We introduce the problem of mining user-defined temporal patterns in association rules. We introduce the notion of using a *calendar algebra* to describe phenomena of interest in association rules. This calendar algebra is used to define and manipulate groups of time intervals. This is an important and novel contribution because real-life patterns can be described succinctly using simple algebraic expressions. (See below and Section 3 for examples.)
- We introduce the notion of finding *fuzzy* patterns in association rules. This notion allows us to find patterns in the data that *approximately* match the user-defined patterns.
- Our techniques extend in a natural way to handle multiple units of time.

The usage of a calendar algebra to specify patterns is of vital importance because the number of possible patterns over a time-interval is exponential in the size of the time interval. In order to simplify the process of writing calendars for an end-user, we also let the user choose from a set of pre-defined calendars. In addition, the users can supply their own calendar expressions. Each such calendar expression corresponds to a *calendar*, which is a collection of time intervals describing some real-life phenomenon.

In order to illustrate the power of calendric expressions, let us consider a stock trading example, where the transactions consist of sets of trades made by people over time. By integrating calendars and association rules, we can discover patterns such as:

- Over the last two years, people have been buying the technology stock QuickRich Software and selling utility stock PowerIsGood Inc. on the days that national employment figures were announced by the government. The US government releases national employment figures on the last day of every month in the year. If the day is a holiday, it is announced the previous business day.
- Selling of technology stock MagicWidget implies buying of MaBell Communications on the days that options expire. (People who bought put options to lock their profits in MagicWidget decide to move their investments to a safer stock when their options expire.) Options expire on the third Friday of certain months. If the day is a holiday, the expiration date is the preceding business day.

Obviously, over the entire set of transactions, neither of these rules might have enough support!

In this paper, we address the problem of describing complicated real-life phenomena as the ones above and finding the association rules and the patterns they follow.

We assume that the transactional data to be analyzed is time-stamped and that time intervals are specified by the user to divide the data into disjoint segments. We believe that users will typically opt for "natural" segmentations of the data based on months, weeks, days, etc., and that users are best qualified to make this decision based on their understanding of the underlying data. Though we primarily concentrate on the case where the user specifies a segmentation based on a single time unit, we believe that our techniques can naturally be extended to the case of hierarchical segmentations. Further, we assume that the user will supply a set of calendar algebra expressions, the calendars corresponding to which are to be detected in the association rules. As mentioned above, the user can also choose from a number of predefined calendar expressions that make the job of supplying the calendars simple. We parse the calendar expressions into calendars, each of which is simply a set of intervals.

We refer to an association rule as *calendric* if the rule has the minimum confidence and support during every time unit contained in a calendar, modulo a mismatch threshold, which allows for a certain amount of error in the matching. This mismatch threshold models the fact that, in real life, the association rule will hold for *most* but not *all* the time units of the calendar. The calendar is then said to *belong* to the rule. The rule need not hold for the entire transactional database, but rather only for transactional data during the time units specified by the calendar (modulo the mismatch threshold). We define the problem of *mining calendric association rules* as the generation of all association rules along with their calendars. Given a large database consisting of transactional information, where each transaction consists of a transaction-id, a set of items and a timestamp, and a set of "interesting" calendar expressions, our goal is to provide efficient algorithms to discover calendric association rules.

Our treatment of calendars is based on the framework developed in [All85, LMF86] and the implementation reported in [CSS94]. We parse algebra expressions using an LALR parser and evaluate calendars as sets of intervals.

For the rule mining, many of the techniques we developed in [ÖRS98] apply in the context of calendric rules, though the actual details are quite different. We first consider a relatively straightforward extension of existing association rule mining techniques for solving this problem. This extension treats association rules and calendars independently. It applies one of the existing methods for discovering association rules to each segment of data and then applies simple pattern matching algorithms to detect calendars in association rules. The *pruning* and *skipping* techniques we developed for cyclic association rules can be applied to calendric association rules also. This allows us to significantly reduce the amount of wasted work performed during the data mining process. We demonstrate the effectiveness of these techniques by presenting the results of a series of experiments.

The remainder of this paper is organized as follows. In Section 2, we define the problem of discovering calendric association rules formally. In Section 3, we describe our algebra for calendars and provide details of our evaluation techniques. In Section 4, we discuss the shortcomings of the existing techniques to discover calendric association rules and present two new techniques to solve this problem. Implementation details of our prototype are described in Section 5. The experimental evaluation of the two techniques is presented in Section 6. Finally, we present our conclusions in Section 7 and identify directions for future research.

2 Problem Definition

Let $\mathcal{I} = \{i_1, i_2, \ldots, i_N\}$ denote a set of *data items*. A transaction is defined to be a subset of $\mathcal{I}$. An itemset is also defined to be a subset of $\mathcal{I}$. We use the letters $X, Y, X_1, Y_1, \ldots$ to denote itemsets. If X and Y are itemsets, then XY represents the set union of X and Y.

Given a set of items $\mathcal{I}$ and a set of transactions $\mathcal{T}$, the problem of discovering association rules is defined as finding relationships between the occurrences of itemsets within transactions. An *association rule* of the form $X \rightarrow Y$ is a relationship between the two disjoint itemsets X and Y. An association rule is described in terms of *support* and *confidence*. The support of an itemset X over the set of transactions $\mathcal{T}$ is the fraction of transactions that contain the itemset. An itemset is called *large*, if its supports exceeds a given threshold sup_{min}. The *confidence* of a rule $X \rightarrow Y$ over a set of transactions $\mathcal{T}$ is the fraction of transactions containing X that also contain Y. The association rule $X \rightarrow Y$ holds, if XY is large and the confidence of the rule exceeds a given threshold con_{min}.

In order to deal with calendric association rules, we enhance the transaction model by a time attribute that describes the time when the transaction was executed. In this paper, we primarily focus on the case where a single unit of time is given (e.g., by the user) since we believe that our techniques extend in a natural way to handle more complicated treatments of time, like a time hierarchy. We denote the j^{th} time unit, $j \geq 0$, by t_j. It corresponds to the time interval $[j \cdot t, (j+1) \cdot t)$, where t is the unit of time. We denote the set of transactions executed in time unit t_j by $\mathcal{T}[j]$.[1]

The support of an itemset X in $\mathcal{T}[j]$ is the fraction of transactions in $\mathcal{T}[j]$ that contain the itemset, whereas the confidence of a rule $X \rightarrow Y$ in $\mathcal{T}[j]$ is the fraction of transactions in $\mathcal{T}[j]$ containing X that also contain Y. An association rule $X \rightarrow Y$ holds in time unit t_j, if the support of XY in $\mathcal{T}[j]$ exceeds sup_{min} and the confidence of $X \rightarrow Y$ exceeds con_{min}.

At a low-level, a *calendar* C is set of (possibly interleaved) time intervals $\{(s_1, e_1), (s_2, e_2), \ldots, (s_k, e_k)\}$. (See Section 3 for a detailed discussion of calendar description and evaluation.) C is said to contain time unit t if it contains an interval (s_j, e_j) such that $s_j \leq t \leq e_j$. We denote the mis-match threshold by m. m is an integer that limits the number of mis-matches that can occur. We say a calendar *belongs* to an association rule $X \rightarrow Y$, if the rule has enough confidence and support for the time units contained in the calendar with at most m mis-matches. In other words, if the calendar contains w time units, the association rule has to hold for at least $w - m$ of them. Similarly, the calendar is said to belong to an itemset X if the support of X exceeds sup_{min} in at least $w - m$ time units.

[1] We will refer to $\mathcal{T}[j]$ specifically as "time segment j" or generically as a "time segment."

Example 2.1 *Let the unit of time be day. Consider the calendar consisting of the days that national unemployment figures were announced by the US government in 1996. The calendar corresponding to those days is* $C = \{(31,31), (60,60), (89,89), (121,121), (152,152), (180,180), (213,213), (243,243), (274,274), (305,305), (334,334), (366,366)\}$. *(We assume in this example that days are numbered consecutively starting with January 1, 1996 as day 1.)*

Let us assume that the mismatch threshold is 0. *If we have a transactional database of trades made by people, we will say that calendar C belongs to the rule "Buying of QuickRich Software → Selling of PowerIsGood Inc.", if the rule has enough support and confidence on days* 31, 60, 89, 121, 152, 180, 213, 243, 274, 305, 334, 366 *of year 1996.*

If, on the other hand, the mis-match threshold is 4, *then the rule "Buying of QuickRich Software → Selling of PowerIsGood Inc." has to hold for at least* $12 - 4 = 8$ *of the* 12 *days in* 31, 60, 89, 121, 152, 180, 213, 243, 274, 305, 334, 366. □

Given a set of transactions and a set of template calendars, we define the problem of discovering calendric association rules as discovering relationships between the presence of items in the transactions that follow the patterns set forth in the calendars.

An association rule can be represented as a binary sequence where the 1's correspond to the time units in which the rule holds and the 0's correspond to the time units in which the rule does not have the minimum confidence or support. For instance, if the binary sequence 001100010101 represents the association rule $X \rightarrow Y$, then $X \rightarrow Y$ holds in $\mathcal{T}[3]$, $\mathcal{T}[4]$, $\mathcal{T}[8]$, $\mathcal{T}[10]$, and $\mathcal{T}[12]$. The calendar $\{(4,4), (8,8), (12,12)\}$, which corresponds to the a cycle of length 4, belongs to the association rule since the association rule is valid on the 4th, 8th and 12th time units. (Unlike variables in programming languages, calendars start from unit one!) Similar to association rules, itemsets can also be represented as binary sequences where 1's correspond to time units in which the corresponding itemset is large and 0's correspond to time units in which the corresponding itemset does not have the minimum support. These binary sequence representations will be useful to prove the correctness of the algorithms that we will develop for discovering calendric association rules.

In the next section, we introduce a simple and powerful calendar algebra for defining and manipulating calendars.

3 Calendar Algebra

Our framework for calendar algebras is based on the work reported in [All85, LMF86] and the implementation reported in [CSS94].

There are two main components to our framework. One is the definition and use of calendar algebras to define and manipulate sets of time intervals. The other is the use of a *calendric system* to define a basic calendric framework like the Gregorian or the Jewish calendar. To the best of our knowledge, the work reported in [CSS94], which itself is based quite closely on the framework developed in [All85, LMF86], is the only one that deals with calendar algebras. The work of Snodgrass and his colleagues, for example [SSD+92], has concentrated primarily on the latter problem of developing a calendric framework and on the problem of integrating time into database models and SQL. Hence, this work is not relevant to the first issue of calendar algebras. However, it is very relevant to the second issue of defining the basic calendric framework. We chose to follow the model developed in [All85, LMF86] for this problem because it is more convenient to implement. However, the work on calendar algebras in this section can be applied to *any* system that defines and implements basic calendars.

Following [All85, LMF86], we define a *calendar* to be a structured collection of intervals. Let $s_1, s_2, \ldots, s_k, e_1, e_2, \ldots, e_k$ be integers. We define a collection $S = \{(s_1, e_1), (s_2, e_2), \ldots, (s_k, e_k)\}$, to be a calendar of order 1. A calendar of order 2 is a collection of calendars of order 1 and so on.

In order to capture relationships between two intervals, [All85] defines the following *interval operators* that operate on two intervals (denoted by $\text{int}_1 = (s_1, e_1)$ and $\text{int}_2 = (s_2, e_2)$) and return a boolean value:

- int_1 overlaps $\text{int}_2 \equiv ((s_1 \leq s_2 \leq e_1) \vee (s_2 \leq s_1 \leq e_2))$
- int_1 during $\text{int}_2 \equiv ((s_1 \geq s_2) \wedge (e_1 \leq e_2))$
- int_1 meets $\text{int}_2 \equiv (e_1 = s_2)$
- $\text{int}_1 < \text{int}_2 \equiv (e_1 \leq s_2)$
- $\text{int}_1 \leq \text{int}_2 \equiv ((s_1 \leq s_2) \wedge (e_1 \leq e_2))$

We are now in a position to define the operators of the calendar algebra.

Dicing Operations: For each interval operator, [LMF86] defines two *dicing* operators. These operators work in two modes: (1) they can take an order 1 calendar as their left argument, an interval as their right argument and produce an order 1 calendar as their output; (2) they can take an order 1 calendar as their left argument, an order 1 calendar as their right argument and produce an order 2 calendar as their output. (They produce an order 1 calendar for each interval in their right argument.)

For each interval operator R, [LMF86] defines two dicing operators: *strict*, denoted by $:R:$, and *relaxed*, denoted by $.R.$ If C is an order 1 calendar and c' is an interval, then the two operators are defined as:

$$\begin{aligned} C:R:c' &\equiv \{c \cap c' \mid c \in C \wedge c\,R\,c'\}/\{\epsilon\} \\ C.R.c' &\equiv \{c \mid c \in C \wedge c\,R\,c'\}/\{\epsilon\} \end{aligned}$$

The intersection ($\cap$) between two intervals (s_1, e_1) and (s_2, e_2) is defined as $(\max(s_1, s_2), \min(e_1, e_2))$ and ϵ denotes the interval $(-\infty, \infty)$ that is to be excluded from the result. The definitions for operators that take a calendar as their right hand argument is similar (C' is an order 1 calendar.):

$$\begin{aligned} C:R:C' &\equiv \{\{c \cap c' \mid c \in C \wedge c\,R\,c'\}/\{\epsilon\} \mid c' \in C'\} \\ C.R.C' &\equiv \{\{c \mid c \in C \wedge c\,R\,c'\}/\{\epsilon\} \mid c' \in C'\} \end{aligned}$$

Example 3.1 *Let* WeeksInJan96 *denote the calendar* $\{(-3,4), (5,11), (12,18), (19,25), (26,32)\}$. *Let* JanIn1996 *denote the calendar* $\{(1,31)\}$. *The expression* $WeeksInJan96 : overlaps : JanIn1996$, *which uses the strict operator returns a single order 2 calendar* $\{\{(1,4), (5,11), (12,18), (19,25), (26,31)\}\}$. *Because of the intersection with the interval from the right hand side, the result consists of* only *the portion of the weeks that fall in the interval* $(1,31)$. *The expression* $WeeksInJan96.overlaps.MonthsIn1996$, *which uses the relaxed operator, returns the calendar* $\{\{(-3,4), (5,11), (12,18), (19,25), (26,32)\}\}$. *In this case, every week that overlaps with* $(1,31)$ *is returned in its entirety.* □

Slicing Operations: Let C be a calendar and p an integer. Two slicing operators denoted by $(p)/C$ and $[p]/C$ operate on C and replace each of the order 1 collections contained in C with the result of the slicing operation. The operator $(p)/C$ replaces each order 1 calendar in C with its p^{th} element and returns the result. For example, while operating on an order 1 calendar, $(p)/C$ simply returns the p^{th} interval in C. The operator $[p]/C$ replaces every order 1 calendar with a calendar consisting of the p^{th} element. For example, while operating on an order 1 calendar, $[p]/C$ returns a calendar consisting of the p^{th} element. If p is negative, indexing is done from the end of the calendar. For example, $(-1)/C$ returns the last element of C. Finally, instead of a single integer p, one is allowed to specify a list of integers for the slicing operation. $[p_1, p_2, \ldots, p_k]/C$ replaces each order 1 calendar with a calendar consisting of the p_1^{th}, p_2^{th}, etc. elements while $(p_1, p_2, \ldots, p_k)/C$ replaces each order 1 calendar with the p_1^{th}, p_2^{th}, etc. elements.

Additional Operations: In addition to the operations defined above, we define and use the minus ($-$) and the plus ($+$) with their usual set-theoretic meanings on calendars. We also use a *flatten* operator which takes an order k calendar and produces an order $k-1$ calendar which is a single calendar made of the all elements of the constituent order $(k-1)$ calendars.

Example 3.2 *Let* WeeksInJan96 *denote the calendar* $\{(-3,4), (5,11), (12,18), (19,25), (26,32)\}$. *The expression* $[3]/WeeksInJan96$ *returns the calendar* $\{(12,18)\}$. *The expression* $[-2]/WeeksInJan96$ *returns the calendar* $\{(19,25)\}$, *while the expression* $[3,4]/WeeksInJan96$ *returns the calendar* $\{(12,18), (19,25)\}$.

The expression $flatten\{\{(-3,4), (5,11), (12,18), (19,25), (26,32)\}\}$ *returns* $\{(-3,4), (5,11), (12,18),$

$(19,25)$, $(26,32)\}$, *while the expression* $flatten\{\{(1,1)\}, \{(5,5)\}\}$ *returns* $\{(1,1), (5,5)\}$. □

The operators we have introduced thus far simply operate on calendars. In fact, one can see that they are quite reminiscent of nested relational algebra [RKS88]. In order to be able to define real-life calendar expressions, one needs a *calendric* system like the Gregorian calendar system. Following [CSS94], we introduce the Gregorian calendar by defining what are called *basic* calendars. They are SECONDS, MINUTES, HOURS, DAYS, WEEKS, MONTHS, YEARS, DECADES, and CENTURY, and refer to the corresponding familiar temporal concepts. In addition, we use a reference point in time called the *origin* of the calendric system. Our origin is the UNIX system start data, Jan 1, 1970 and this is taken to be the starting point for all the basic calendars.

Relationships between basic calendars are kept in a table with the following structure:

CALTABLE(cal1: string, cal2: string,
repList : array of integers, offset : integer)

In CALTABLE, cal1 and cal2 are one of the basic calendars. For example, an entry {YEARS, MONTHS, 12, 0} expresses the relationship that each year is made up of 12 months. To express something more complicated like the relationship between years and days, an entry of the form {YEARS, DAYS, (365, 365, 366, 365) , 0 } is used. This means that the first year from the origin (1970)[2] has 365 days, and that the second year from the origin also has 365 days. The third year, being a leap year, has 366 days. The fourth year has 365 days. After this, the pattern repeats over. (Obviously, this doesn't handle leap centuries. To handle this, a more complicated expression is needed). The "offset" is used to take care of the basic calendars whose boundaries do not match with the chosen origin. Thus an entry of the form {WEEKS, DAYS, 7, 4} is used to take into consideration the fact that January 1, 1970 lies on a Thursday (assuming that a week begins on a Monday).

Once the relationships between the basic calendars are defined, we can easily define fairly complicated temporal expressions. We show an example below. Additional examples can be found in [RMS98].

Example 3.3 *Mondays that overlap the first day of a month are expressed by the calendar algebra expression:*

flatten (((1) / (DAYS :during: WEEKS))
:during:
((1) / (DAYS :during: MONTHS)))

The (DAYS :during: WEEKS) expression expresses weeks in terms of its constituent days. The (1)/(DAYS :during: WEEKS) then selects the first day of every week, producing a calendar consisting of the first day of every week. Similarly, the expression (1)/(DAYS :during: MONTHS) returns a calendar consisting of the first day of every month. The "during" between these calendars returns an order two calendar consisting of the Mondays that occur during the first days of the months. The flatten reduces this to an order 1 calendar containing the result. □

[2] As alluded to before, all calendric systems are indexed from 1, rather than 0. Also, an interval over time is assumed to never contain 0. For example, the interval (-3, 1) contains the time units -3, -2, -1, and 1, but not 0.

Handling Multiple Time Granularities: The techniques introduced in this section are general enough to handle multiple granularities; they only rely on being able to express higher granularities in terms of lower granularities—for example, months in terms of days. When dealing with multiple calendar algebra expressions, it becomes necessary to express them in terms of a common time granularity. For example, if dealing with calendars that talk about weeks and months, it is simpler to express them in terms of days. Such a time unit is always guaranteed to exist in any calendric system and corresponds to the notion of a *chronon* in temporal database literature [SSD$^+$92].

We have implemented a parser and an evaluator to parse and evaluate the calendar algebra efficiently. Our implementation loosely follows the implementation described in [CSS94]. The parser is an LALR parser. Starting and ending points and the granularity for the output calendar are supplied by the user or when possible, deduced from the algebra expressions. The result of the evaluation is an order 1 calendar (a collection of intervals), which is then passed to the data mining routines. Due to lack to space, we omit a detailed description here.

4 Discovering Calendric Association Rules

As discussed in [ÖRS98], existing algorithms for discovering association rules cannot be applied directly to discover cyclic association rules. It follows then that these algorithms cannot be applied to solve the problem of calendric association rules, which offer far more expressive power than cyclic association rules.

For example, extending the attributes of an association rule in order to capture rules like $(day = Monday) \cup X \rightarrow Y$ is clearly infeasible because there are an *exponential* number of calendars, 2^t, that are possible in a time period of length t.

4.1 The Sequential Algorithm

As with cyclic rules, the straight-forward approach to discovering calendric association rules is to treat the problem of calendar detection and association rule mining separately. That is, we generate the rules in each time unit with one of the existing methods [AS94, SON95] and then apply a pattern matching algorithm (See Section 4.4) to discover calendars. We refer to this approach as the *sequential* algorithm.

Existing algorithms discover association rules in two steps. In the first step, large itemsets are generated. In the second step, association rules are generated from the large itemsets. The running time for generating large itemsets can be substantial, since calculating the supports of itemsets and detecting all the large itemsets for each time unit grows exponentially in the size of the large itemsets. To reduce the search space for the large itemsets, the existing algorithms exploit the following property:

"Any superset of a small itemset must also be small."

The existing algorithms calculate support for itemsets iteratively and they prune all the supersets of a small itemset during the consecutive iterations. Let us refer to this pruning technique as *support-pruning*. In general, these algorithms execute a variant of the following steps in the k^{th} iteration:

1. The set of candidate $k-$itemsets is generated by extending the large $(k-1)-$itemsets discovered in the previous iteration (support-pruning).
2. Supports for the candidate $k-$itemsets are determined by scanning the database.
3. The candidate $k-$itemsets that do not have minimum support are discarded and the remaining ones constitute the large $k-$itemsets.

The idea is to discard most of the small $k-$itemsets during the support-pruning step so that the database is searched only for a small set of candidates for large $k-$itemsets.

In the second step, the rules that exceed the confidence threshold con_{min} are constructed from the large itemsets generated in the first step with one of the existing algorithms. For our experimental evaluation of the sequential algorithm, we implemented the apriori and the ap-genrules algorithms from [AS94]. Once the rules of all the time units have been discovered, calendars that belong to the rules need to be detected. Let r be the number of rules detected and k be the number of time units a calendar contains. Checking to see whether the calendar belongs to the rules can be done in time $O(r \times k)$.

However, in practice, it turns out that the number of association rules is substantially more than the number of large itemsets discovered. This typically cause the sequential algorithm to run out of real memory causing it to perform many I/O's to bring relevant portions of the data into memory. In particular, as the average itemset size increases, this becomes a severe problem for the sequential algorithm.

A better approach is to discover the large itemsets over all the time units and then to use these to discover the association rules and their associated calendars. This speeds up the sequential algorithm substantially. The details of this optimization are given in Section 6. Of course, the disadvantage of this optimization is that one can no longer treat the association rule mining module as a "black box."

4.2 Pruning, Skipping and Elimination

The major portion of the running time of the sequential algorithm is spent to calculate the support for itemsets. The three techniques we used in the cyclic association rules paper to reduce the number of itemsets for which support must be calculated —*pruning, skipping,* and *elimination*— can be applied in the case of calendric rules, though the details of the application are quite different. These techniques rely on the following fact:

"A calendar that belongs to the rule $X \rightarrow Y$ also belongs to the itemset XY."

Therefore, eliminating calendars as early as possible can substantially reduce the running time of calendric association rule detection.

Skipping is a technique for avoiding counting the support of an itemset in time units which are guaranteed to be contained in any calendar that can belong to the itemset. Skipping is based on the following property:

"If time unit t_j is not contained in any calendar that belongs to an itemset X, then there is no need to calculate the support for X in time segment $\mathcal{T}[j]$."

However, this technique can be applied only if we have information about the calendars of an itemset X. But the calendars of an itemset X can be computed exactly only after we compute the support of X in all the time segments! In order to avoid this self-dependency, we try to approximate the calendars of itemsets. To do this, we use a technique we call *pruning*, which is based on the following Lemma, whose proof can be found in [RMS98]:

Lemma 4.1 *If a calendar C belongs to an itemset X then it must also belong to all of X's subsets.*

Therefore, one can arrive at an upper bound on the calendars that belong to an itemset X by looking at the calendars that belong to X's subsets. By doing so, we can reduce the number of potential calendars that belong to X, which, in turn (due to skipping), reduces the number of time units in which we need to calculate support for X. Thus, pruning is a technique for computing the *potential* calendars of an itemset by merging the calendars of the itemset's subsets.

However, it is possible in some cases that we cannot compute the potential calendars of an itemset. For example, when we are dealing with singleton itemsets. In these cases, we need to assume that an itemset X has every possible calendar and therefore, calculate the support for X in each time segment $\mathcal{T}[j]$ (except the time units eliminated via support-pruning). This is, in fact, what the sequential algorithm does.

Example 4.2 *If we know that the calendar $\{(4,4), (8,8), (12,12)\}$ is the only calendar that belongs to items A and B, then pruning implies that the only calendar that can belong to AB is also $\{(4,4), (8,8), (12,12)\}$.*

In turn, skipping implies that we have to calculate the support of AB only in $\mathcal{T}[4]$, $\mathcal{T}[8]$, and $\mathcal{T}[12]$ rather than all the time segments. □

We now introduce one more optimization technique, which we refer to as *elimination*, that can be used to further reduce the number of potential calendars of an itemset X. Elimination is used to eliminate certain calendars from further consideration once we have determined they cannot exist. Elimination relies on the following Lemma, whose proof follows immediately from the definition of how a calendar belongs to an itemset:

Lemma 4.3 *If the support for an itemset X is below the minimum support threshold sup_{min} in m time units contained in a calendar C, where m is the mis-match threshold, then C cannot belong to X.*

Elimination enables us to discard calendars that an itemset X cannot have as soon as possible as demonstrated in the following example.

Example 4.4 *If the mis-match threshold is 0, and we discover that itemset X does not have enough support in $\mathcal{T}[4]$, we know that the calendar $\{(4,4),(8,8),(12,12)\}$ cannot belong to X.* □

4.3 The Interleaved Algorithm

The pruning, skipping and elimination techniques lead us to the *interleaved* algorithm for discovering calendric association rules. The intuition behind the interleaved algorithm is that we will use the calendars associated with itemsets to minimize the number of candidates whose support we need to count. We will also try to minimize the number of potential calendars that need to be associated with itemsets.

The interleaved algorithm consists of two phases. In the first phase, the calendars belonging to large itemsets are discovered. In the second phase, calendric association rules are generated.

In the first phase of the interleaved algorithm, the search space for the large itemsets is reduced using pruning, skipping and elimination. Figure 1 outlines this phase. Note that at the end of Step 2, we know the set of calendars that *actually* belong to itemsets of size k.

Phase One terminates when the list of potential calendars for each $k-$itemset is empty. Pruning, skipping and elimination can reduce the candidate $k-$itemsets for which support will be counted in the database substantially, and therefore can reduce the time needed to calculate large itemsets. This is demonstrated by the following example.

Example 4.5 *Suppose that the only calendar we are interested in is $C = \{(4,4), (8,8), (12,12)\}$ and sequences 1110000000111111111 and 1111010111111111111 represent items A and B, respectively. (Recall from Section 2 that a 1 in such a sequence indicates that the item has enough support and that a 0 indicates that it doesn't.) Assume also that the mis-match threshold is 0.*

If the sequential algorithm is used, then support for A and B will be calculated in all the time segments, and support for AB will be calculated in time segments 1–3, *and* 11–19. *In the interleaved algorithm, support for A will be calculated only in time unit* 4, *at which point C is eliminated from consideration for A. Support for B is calculated in time segments* 4, 8, *and* 12 *and C is found to belong to B. The itemset AB has no potential calendars because A has none and hence support for AB is* never *calculated!* □

In the second phase of the interleaved algorithm, calendric association rules can be calculated using the calendars and the supports of the itemsets without scanning the database. Interleaving calendar detection with large itemset detection also reduces the overhead of rule generation phase. This is because a calendar of the rule $X \to Y$ must belong to the itemset XY, and at the end of the first phase of the interleaved algorithm we already know the calendars of large itemsets. Thus, the set of candidate calendars for a rule $X \to Y$ initially consists of the set of calendars of the itemset XY. As a result, we need to calculate the confidence of a rule $X \to Y$ only for time units that are contained in the calendars belonging to XY. Moreover, we can apply elimination here also. If C is a calendar belonging to XY, and we encounter m time units in which $X \to Y$ does not have minimum confidence *or* XY doesn't have enough support, we can eliminate C from the list of potential calendars for $X \to Y$.

For each k, $k \geq 1$:

1. If $k = 1$, then all possible calendars are initially assumed to exist for each single itemset. Otherwise (if $k > 1$), pruning is applied to generate the potential calendars for $k-$itemsets using the calendars for $(k-1)-$itemsets. If the list of potential calendars for each $k-$itemset is empty, Phase One terminates.
2. Time segments are processed sequentially. For each time unit t_j :
    - 2.1 Skipping determines, from the set of candidate calendars for $k-$itemsets, the set of $k-$itemsets for which support will be calculated in time segment $\mathcal{T}[j]$.
    - 2.2 If a $k-$itemset X chosen in Step 2.1 does not have the minimum support in time segment $\mathcal{T}[j]$, then the mis-match count is incremented by 1 for each potential calendar associated with X that contains t_j. If this mis-match count exceeds the mis-match threshold for a particular calendar, that calendar is eliminated from the list of potential calendars for X.

Figure 1: Phase One of the interleaved algorithm

4.4 Calendar Detection

In order to detect whether a calendar C belongs to an association rule, we need to examine the support and confidence of the rule for every time unit contained in the calendar. If the calendar contains k time units, this can be done in $O(k)$ steps. And in general, this is the best we can do since a calendar can contain arbitrary time units.

The situation is less clear if we have a *set* of calendars that we have to check against a given rule or itemset. In this case, it should be possible to examine the structures of the calendars to eliminate duplicate work. However, we do not address this issue in this paper. The problem is similar to the problem of detecting *all cycles* that belong to a given binary sequence [ÖRS98].

The following lemma shows that the skipping technique in Section 4.2 cannot affect calendar detection.

Lemma 4.6 *In the course of determining whether a calendar C belongs to an association rule (itemset), suppose that C does not contain time unit t. Whether C belongs to the rule (itemset) or not is unaffected by the support and confidence (support) of the association rule (itemset) in time unit t.*

Proof: This follows from the definition of *belongs* that states that a calendar belongs to a rule (itemset) if the rule has enough support and confidence (support) for every time unit that is contained in the calendar (modulo the mismatch threshold). The support and confidence (support) of a rule (itemset) in a time unit t not belonging to the calendar, then clearly does not affect the determination process. □

4.5 Proof of Correctness

We tie everything together in this section by formally showing that the interleaved and sequential algorithm are equivalent in that they discover the same calendric association rules. We do this in two steps: First, we prove that every calendar belonging to every large itemset in the input data is discovered correctly by the interleaved algorithm. We then prove that the calendric association rules discovered by the interleaved algorithm are the same as those discovered by the sequential algorithm and vice-versa. Due to lack to space, we omit the proofs of the Lemma and the Theorem. They can be found in [RMS98].

Lemma 4.7 *Phase One of the interleaved algorithm correctly computes every calendar belonging to every itemset.*

Theorem 4.8 *A calendric association rule is detected by the interleaved algorithm iff it is detected by the sequential algorithm.*

The Lemma above is an induction on the size of the itemsets. Proof of the Theorem proceeds in two parts: To show that the interleaved algorithm computes every calendric association rule that the sequential algorithm does and vice-versa.

4.6 Multiple Granularities and Further Optimizations

We point out that all the results in this section apply directly even when we are handling multiple time granularities. As long as the different granularities are expressed in terms of a common time unit, one only has to modify the definition of the technical terms "contains" and "belongs" for the proofs to apply for multiple granularities.

Multiple granularities also introduce the possibility of further optimizations when some calendars strictly subsume other calendars. For example, it might be the case that every time unit t_j that belongs to a calendar C_1 also belongs to another calendar C_2. In such cases one can cut down on the number of calendars searched for by organizing the calendars into a calendar hierarchy based on the subsumption relationship. However, experimental results indicate two factors that make such optimizations less than useful in practice: (a) Since calendars are arbitrary user-defined expressions, it is not very likely that users will define and look for two calendars that subsume each other; (b) The running time of calendric association rule discovery is completely dominated (by a factor of over 100) by the time to count support for large itemsets. Improvements in optimizing the search for calendars that take advantage of inter-relationships between calendars are unlikely to yield much gains in the overall running time.

D	Number of transactions/time segment
T	Avg. size of transactions
I	Avg. size of the maximal potentially large itemsets
L	Number of maximal potential large itemsets
N	Number of items

Table 1: Parameters for data generation from[AS94, AMS+96]

5 Implementation Details

In this section, we present the implementation details of the prototype that we built for discovering calendric association rules. We first describe the synthetic data generator used to generate our data.

5.1 Data Generation

Our data generator is based on the synthetic data generator used in [AS94, AMS+96]. We augmented it to generate data for calendric association rules. In addition to the parameters used by [AS94, AMS+96] shown in Table 1, we used additional parameters shown in Table 2. (The parameters are described in the following paragraphs.)

The generation of the large itemsets and their weights closely follows the procedure in [AS94, AMS+96]. We generate L itemsets of average size I. Each itemset is associated with a weight, which is an exponentially distributed random variable. Each itemset has, on the average, C_{item} calendars that belong to it which means that the itemset is used for data generation during any time unit contained in any of its calendars.

In order to model the fact that real world data will consist of a mixture of calendric rules and non-calendric rules,

u	Number of time units of data generated
C_{item}	Avg. number of calendars associated with each large itemset
ν	Avg. level of "noise" in the data generated

Table 2: New parameters for calendric association rule generation

Number of transactions/time segment, D	10000
Number of items, N	1000
Avg. size of large itemsets, I	4
Number of large itemsets, L	1000
Avg. transaction size, T	10
Number of time units, u	600
Avg. number of calendars per itemset, C_{item}	3
Avg. level of "noise" in the data generated, ν	0.3

Table 3: Default settings for parameters in data generation.

we use the "noise" parameter ν, which is a real number between 0 and 1. In a particular time unit, a large itemset is "active" (in the sense that transactions in that time unit will contain that itemset) independent of the calendars that belong to it with a probability ν.

The calendars themselves are supplied as files containing the algebra expressions. The data generation program reads these files and parses the expressions into calendars which are then used for data generation.

At the beginning of each time unit, the data generation algorithm first determines which large itemsets should be used for data generation in that time unit. This is done by checking to see if the current time t is contained in any of the calendars that belong to it. Following this, a determination is made as to whether the noise parameter dictates that the itemset be used. Once this is done, the weights associated with the large itemsets determine their occurrences in the transactions for the time unit.

The default values we used for the parameters in our experiments are shown in Table 3. We conducted individual sets of experiments that varied these parameters. We describe the variations when we describe the individual experiments.

When the parameters are set to the above default values, the size of the data generated is about 150 megabytes (MB) for all the time units combined.

5.2 Prototype Implementation Details

We use the apriori algorithm from [AS94, AMS+96] as our basic data mining algorithm. The sequential algorithm is based directly on apriori, with optimizations to speed up the counting of support of itemsets of size 2. We use an array for this instead of a hash-tree when memory permits: We found the array to be a much faster technique for discovering 2-itemsets.

The interleaved algorithm uses a hash-tree, described in [AS94, AMS+96], to store the large itemsets, their patterns and support counts. In addition, during the processing of an individual time segment, the interleaved algorithm uses a temporary hash-tree as well. Candidate generation (generation of itemsets of size $k+1$ and their candidate calendars from itemsets of size k) is based on pruning. Figure 2 outlines the first phase of the *interleaved* algorithm (detection of calendars belonging to itemsets) in pseudocode.

As referred to before in Section 4.1, the number of association rules with calendars is typically much larger than the number of itemsets with calendars. If the sequential algorithm is applied naively to discover all the rules and their calendars, it consumes enormous amounts of memory to store the status of *all the association rules for all the time units.* We found out that the sequential algorithm performs significantly better if it is used to only find large itemsets. After this, calendars belonging to the large itemsets can be discovered. At this point, the same procedure used to generate rules for the interleaved algorithm can be applied to the sequential algorithm as well. Accordingly, in our experimental comparison, we only compared the times needed by the two algorithms to generate the calendars belonging to the large itemsets. For rule generation, a simple variant of the procedure *Level_GenRuleCycles* in [ÖRS98] can be used.

```
/* This algorithm uses two hash-trees. itemset-hash-tree
contains candidates of size k, their potential calendars,
and space to store support counts for the relevant time
units. If a calendar contains time unit t and belongs (or
potentially can belong) to an itemset, that itemset is said
to be "active" at time unit t. tmp-hash-tree, during the
processing of time segment t, contains all the itemsets
that are active in t. */ initially, itemset-hash-tree contains
singleton itemsets and all possible calendars

$k = 1$
while (there are still candidates in itemset-hash-tree
    with potential calendars)
  for $t = 1$ to $u$
    insert "active" itemsets from itemset-hash-tree
      into tmp-hash-tree // skipping
    measure support in current time segment for
      each itemset in tmp-hash-tree
    forall $l \in$ tmp-hash-tree
      if ($sup_l < sup_{min}$)
      then
        increment mis-match count for every calendar
          (potentially) belonging to $l$ that contained $t$.
        if mis-match count exceeds threshold for a
          particular calendar $C$, delete it from $l$'s list
          of potential calendars // elimination
      else insert ($l$, $sup_l$, $t$) into itemset-hash-tree
      // this just inserts a ($sup_l$, time) entry in one of
        itemset $l$'s fields
    end forall
    empty tmp-hash-tree
  endfor
  generate new candidates of size $k + 1$ using pruning
  $k = k + 1$
  empty itemset-hash-tree after copying it
  insert new candidates into itemset-hash-tree
endwhile
```

Figure 2: The *interleaved* algorithm for detection of calendars belonging to itemsets.

We conducted our experiments on a lightly loaded Sun Sparc 20 machine with 64 MB of memory running Solaris

2.5.1. A Seagate 9 GB SCSI disk was used for our experiments. The hard disk had a streaming read throughput of about 5 megabytes/sec (MBps) and a streaming write throughput of about 4 MBps. Since our experiments were CPU bound most of the time, we only report wall clock times for the various experiments.

Due to space limitations, we refer the reader to [RMS98] for a detailed discussion of memory management issues for the interleaved and sequential algorithms.

5.3 Handling Multiple Granularities

Multiple time units can be seamlessly integrated in our approach. As discussed before, calendars over multiple time units can be expressed in terms of a common time unit that is guaranteed to exist. For example, calendars over months and weeks in terms of days. Once this is done, the sequential algorithm can run a copy of itself for each granularity simultaneously (to avoid multiple scans of data). The interleaved algorithm can keep track of itemsets and their calendars of different granularities easily. A detailed algorithm is given in [RMS98].

6 Experimental Results

We now present the results of an extensive set of experiments conducted to analyze the behavior of the sequential and interleaved algorithms. We only present results for the behavior of the algorithms while handling a single time granularity. We believe that the results will not be seriously affected by the presence of multiple time granularities.

6.1 Dependence on Minimum Support

In these experiments, it should be kept in mind that the sequential algorithm had an inherent advantage in considering the time segments of the data one by one. All but the largest time segments that we used fit entirely in the main memory of the machine that we used. The interleaved algorithm, which sweeps repeatedly through the entire data, incurs more I/O's. It should also be kept in mind that the sequential algorithm used in the experiments uses the optimization mentioned in Section 4.1 and only computes large

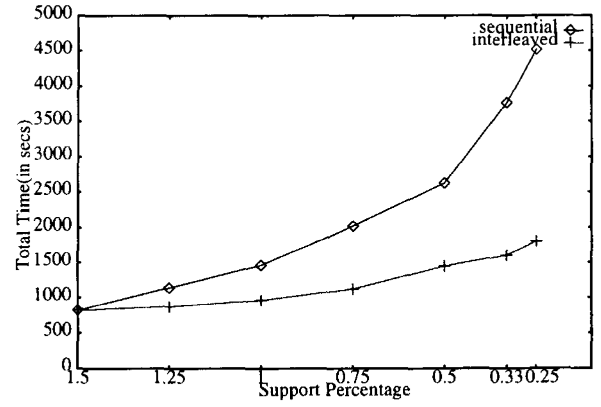

Figure 3: Execution time plotted against support for the sequential and interleaved algorithms.

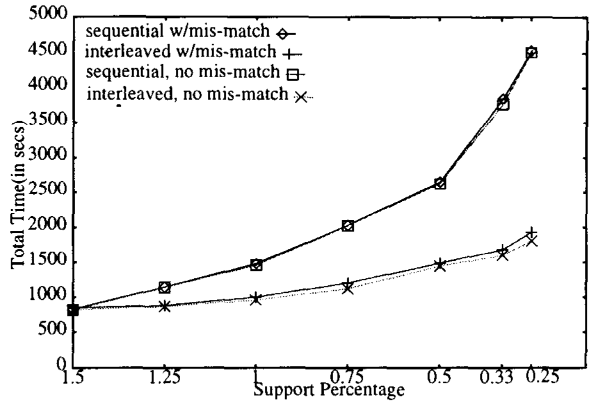

Figure 4: Execution time at various levels of support for sequential and interleaved algorithms with and without mismatches.

itemsets and their calendars. Using the sequential algorithm to compute the rules and their calendars seriously undermines the performance of the sequential algorithm. Hence, that is not done in the experiments.

Unless other wise indicated, the default value for support used is 0.75%.

Figure 3 plots the execution time for the interleaved and sequential algorithms as support is varied from 1.5% to 0.25%. In these experiments, the mis-match threshold is set to 0 (which implies exact calendar matching). With support set to 1.5%, the executions times of the two algorithms are practically identical. This is because both the interleaved and sequential algorithms do not have much to discover by way of calendric rules because the support is too high. (Since we generate 1000 large itemsets, the average support per itemset is only 0.1%.) As support decreases, the amount of wasted work done by the sequential algorithm increases significantly as the number of itemsets found to be large by the sequential algorithm increases. The interleaved algorithm benefits from its pruning techniques, especially elimination, and this dramatically reduces the number of candidate itemsets for which support must be counted. At support set to 0.25%, the sequential algorithm takes close to 250% the time taken by the interleaved algorithm.

6.2 Calendar Detection with Mis-Matches

Figure 4 shows the running times of the two algorithms with the mis-match threshold set to 2. Neither the sequential or the interleaved algorithm is affected by the presence of mis-matches. To illustrate this, we have superimposed the graph from Figure 3 on Figure 4. The two algorithms are not affected much by mis-matches because counting the support of candidate itemsets dominates the overall running time of both the algorithms. While the presence of mis-matches does undermine the ability of the interleaved algorithm to perform elimination quickly, it is not serious enough to significantly affect the running time.

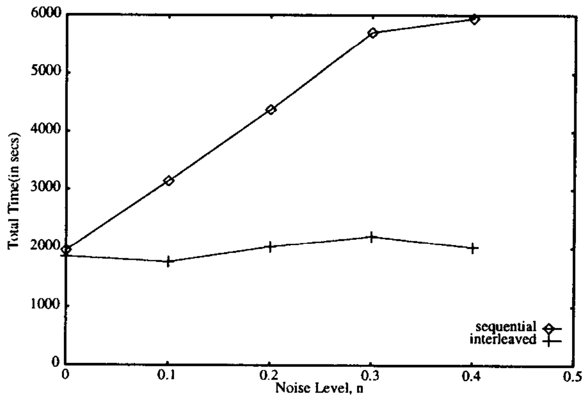

Figure 5: Execution time vs. noise level for the two algorithms at varying levels of support. In the experiments for this graph, support was set to 0.25%.

6.3 Varying Noise Levels

Figure 5 shows the dependence of the two algorithms on noise. When there is no noise, the set of candidates considered by the two algorithms are *exactly* the same. This is because only largesets with calendars occur in the database and both the algorithms have to count support for all of them. As the amount of noise increases, two effects come into play: (1) The amount of wasted work performed by the sequential algorithm increases dramatically since it begins counting support for many itemsets for whom no calendars belong. (2) Noise increases the *maximum* size of large itemsets in the database. This is because it randomly activates large itemsets, and the size of some of these large itemsets is bigger than that of itemsets with calendars. As expected, the interleaved algorithm is completely unaffected by noise. The combination of pruning and elimination obliterates the effects of noise.

6.4 Varying Itemset Size

Figure 6 compares the running time of the two algorithms when the average size of the large itemsets is varied from 3 to 7. Both algorithms are fairly dependent on the *maximum* size of the large itemsets. (This increases the number

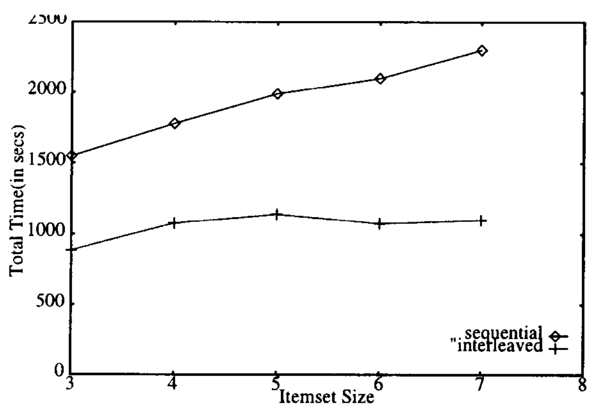

Figure 6: Execution time vs. large itemset size for the two algorithms

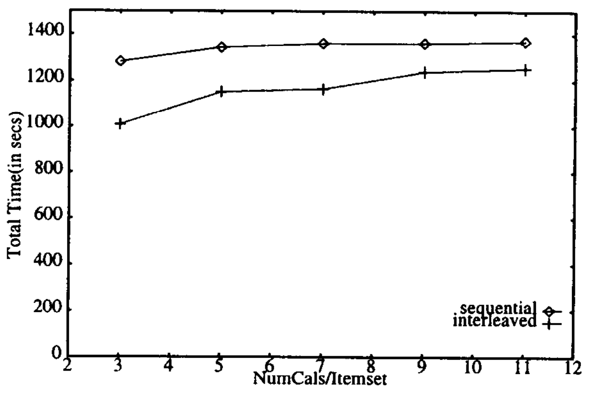

Figure 7: Execution time vs. avg. number of calendars associated with large itemsets.

of passes that they have to make.) Since we control only the *average* size of the large itemsets in our experiments, both algorithms exhibit some non-monotonic behavior as the average large itemset size is increased. The sequential algorithm is slower than the interleaved algorithm by nearly by 75% even at an average itemset size of 3. The gap continues to grow and exceeds 100% as the average size is increased.

6.5 Varying the number of Calendars Associated with a Large Itemset

Figure 7 compares the two algorithms as the average number of calendars associated with large itemsets is varied from 3 to 11. Both the algorithms are only marginally affected by this variation. This indicates that calendar detection is not a significant bottleneck in either of the algorithms.

6.6 Data Size Scaleup

Figure 8 shows the running time of the two algorithms as the time segment size is increased from $10k$ transactions

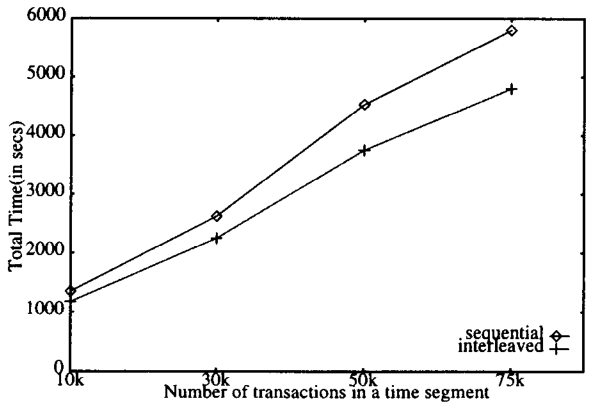

Figure 8: Execution time for the interleaved algorithm as the data size for a single time segment increases from $10k$ transactions to $70k$ transactions. (This corresponds to increasing the total database size from 155 megabytes to 1.1 gigabytes.)

to $70k$ transactions. (The database size increased from 155 megabytes to 1.1 gigabytes.) Both the algorithms can handle large amounts of data fairly well and exhibit good scaleup. The interleaved algorithm continues to exhibit a clear performance superiority throughout.

6.7 Experimental Conclusions

Through a series of experiments, we have demonstrated that detecting calendric association rules can be done efficiently and quickly. The interleaved algorithm outperforms the sequential algorithm convincingly in all the dimensions of the problem. The performance of the interleaved algorithm is sometimes 2 or 3 times faster than that of the sequential algorithm. Hence, we can definitely say that calendric association rule detection benefits significantly from the optimization techniques that we have presented in this paper.

7 Conclusions and Future Directions

In this paper, we have studied the problem of discovering interesting patterns in the variation of association rules over time. Information about such variations will allow analysts to better identify trends in association rules and help better forecasting. By studying the interaction between large itemset detection and calendars, we devised a series of optimization techniques that significantly speed up the discovery of calendric association rules. These optimization techniques allow us to obtain performance benefits ranging from 5% to 250% over a less sophisticated approach.

In future work, we would like to handle the issue of time in other data mining problems like classification where they seem to fit naturally. It would also be interesting to devise online and incremental algorithms for these problems.

Acknowledgments: The authors would like to thank Banu Özden for helpful discussions.

References

[AIS93] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. "Mining Association Rules between Sets of Items in Large Databases". In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, pages 207—216, Washington,DC, May 1993.

[All85] J. F. Allen. "Maintaining Knowledge about Temporal Intervals". In *"Readings in Knowledge Representation"*, pages 509–521. Morgan-Kaufman Publishers, Inc., 1985.

[AMS$^+$96] R. Agrawal, H. Mannila, R. Srikant, H. Toivonen, and A. I. Verkamo. "Fast Discovery of Association Rules". In *"Advances in Knowledge Discovery and Data Mining"*, U. M. Fayyad G. Piatetsky-Shapiro, P. Smyth and R. Uthurusamy (Eds.), pages 307–328. AAAI Press / The MIT Press, 1996.

[AS94] R. Agrawal and R. Srikant. "Fast Algorithms for Mining Association Rules in Large Databases". In *Proceedings of the 20th International Conference on Very Large Data Bases*, pages 487–499, Santiago, Chile, September 1994.

[CSS94] Rakesh Chandra, Arie Segev, and Michael Stonebraker. "Implementing Calendars and Temporal Rules in Next Generation Databases". In *Proceedings of the Tenth International Conference on Data Engineering*, pages 264–273, Houston, Texas, February 1994.

[FMMT96] T. Fukuda, Y. Morimoto, S. Morishita, and T. Tokuyama. "Data Mining Using Two-Dimensional Optimized Association Rules". In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 13–23, Montreal, Canada, June 1996.

[HF95] J. Han and Y. Fu. "Discovery of Multi-level Association Rules From Large Databases". In *Proceedings of the 21st International Conference on Very Large Data Bases*, pages 420–431, Zurich, Switzerland, September 1995.

[LMF86] B. Leban, D. McDonald, and D. Forster. "A Representation for Collections of Temporal Intervals". In *Proceedings of the AAAI-1986 5th Int. Conf. on Artificial Intelligence*, pages 367–371, 1986.

[ÖRS98] Banu Özden, Sridhar Ramaswamy, and Abraham Silberschatz. "Cyclic Association Rules". In *Proceedings of the Fourteenth International Conference on Data Engineering*, February 1998. To appear.

[PCY95] J. S. Park, M. Chen, and P. Yu. "An Effective Hash-based Algorithm for Mining Association Rules". In *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 175–186, May 1995.

[RKS88] Mark A. Roth, Henry F. Korth, and Abraham Silberschatz. Extended algebra and calculus for nested relational databases. *ACM Transactions on Database Systems*, 13(4):389–417, December 1988.

[RMS98] S. Ramaswamy, S. Mahajan, and A. Silberschatz. On the discovery of interesting patterns in association rules. Technical report, Bell Labs, 1998. Available upon request.

[SA95] R. Srikant and R. Agrawal. "Mining Generalized Association Rules". In *Proceedings of the 21st International Conference on Very Large Data Bases*, pages 407–419, Zurich, Swizerland, September 1995.

[SA96] R. Srikant and R. Agrawal. "Mining Quantitative Association Rules". In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 1–12, Montreal, Canada, June 1996.

[SON95] A. Savasere, E. Omiecinski, and S. Navathe. "An Efficient Algorithm for Mining Association Rules in Large Databases". In *Proceedings of the 21st International Conference on Very Large Data Bases*, pages 432–444, Zurich, Swizerland, September 1995.

[SSD$^+$92] M. D. Soo, R. Snodgrass, C. Dyreson, C. S. Jensen, and N. Kline. "Architecural Extensions to Support Multiple Calendars. Technical Report TempIS TR-32, Computer Science Department, University of Arizona., May 1992.

[Toi96] H. Toivonen. " Sampling Large Databases for Association Rules". In *Proceedings of the 22nd International Conference on Very Large Data Bases*, Bombay, India, September 1996.