Discovering Calendar-based Temporal Association Rules*

Yingjiu Li Peng Ning X. Sean Wang Sushil Jajodia
Center for Secure Information Systems
George Mason University, Fairfax, VA 22030, USA
{yli, pning, xywang, jajodia}@ise.gmu.edu

Abstract

A temporal association rule is an association rule that holds during specific time intervals. An example is that eggs and coffee are frequently sold together in morning hours. This paper studies temporal association rules during the time intervals specified by user-given calendar schemas. Generally, the use of calendar schemas makes the discovered temporal association rules easier to understand. An example of calendar schema is (year, month, day), which yields a set of calendar-based patterns of the form $\langle d_3, d_2, d_1\rangle$, where each d_i is either an integer or the symbol $*$. For example, $\langle 2000, *, 16\rangle$ is such a pattern, which corresponds to the time intervals, each consisting of the 16th day of a month in year 2000. This paper defines two types of temporal association rules: precise-match association rules that require the association rule hold during every interval, and fuzzy-match ones that require the association rule hold during most of these intervals. The paper extends the well-known Apriori algorithm, and also develops two optimization techniques to take advantage of the special properties of the calendar-based patterns. The experiments show that the algorithms and optimization techniques are effective.

1. Introduction

Among various types of data mining applications, the analysis of transactional data has been considered important. The notion of *association rule* was proposed to capture the cooccurrence of items in transactions [1]. For example, given a database of orders (transactions) placed in a restaurant, we may have an association rule of the form *egg*$\rightarrow$ *coffee* (support: 3%, confidence: 80%), which means that 3% of all transactions contain the items *egg* and *coffee*, and 80% of the transactions that have the item *egg* also have the item *coffee* in them. The two percentage parameters are referred to as *support* and *confidence* respectively.

An interesting extension to association rules is to include a temporal dimension. For example, if we look at a database of transactions in a supermarket, we may find that *turkey* and *pumpkin pie* are seldom sold together. However, if we only look at the transactions in the week before Thanksgiving, we may discover that most transactions contain *turkey* and *pumpkin pie*, i.e., the association rule "*turkey* $\rightarrow$ *pumpkin pie*" has a high support and a high confidence in the transactions that happen in the week before Thanksgiving.

The above suggests that we may discover different association rules if different time intervals are considered. Some association rules may hold during some time intervals but not during others. Discovering temporal intervals as well as the association rules that hold during the time intervals may lead to useful information. For example, by considering each IP packet in a computer network as a transaction and the attributes in the IP header as items in the transaction, we can use temporal association rules to represent normal network activities in different time periods of a day; attacks to the network may be flagged when the network behaves differently from its normal behaviors.

Informally, we refer to the association rules along with their temporal intervals as *temporal association rules*. In this paper, we propose to use *calendar schemas* as frameworks to discover temporal association rules. A calendar schema is determined by a hierarchy of calendar units. For example, a calendar schema can be (*year*, *month*, *day*). A calendar schema defines a set of *simple calendar-based patterns* (or *calendar patterns* for short). For example, given the above calendar schema, we will have calendar patterns such as *every day of January of 1999* and *every 16th day of January of every year*. Basically, a calendar pattern is formed for a calendar schema by fixing some of the calendar units to specific numbers while leaving other units "free" (so it's read as "every"). It is clear that each calendar pattern defines a set of time intervals.

We assume that the transactions are timestamped so we can decide if a transaction happens during a specific time

*The work was partially supported by ARO under contract number DAAG-55-98-1-0302. Work of Wang was also partially supported by the NSF Career award 9875114.

interval. Given a set of transactions and a calendar schema, our first interest is to discover all pairs of association rule and calendar pattern such that for each pair (r, e), the association rule r satisfies the minimum support and confidence constraint among all the transactions that happen during each time interval given by the calendar pattern e. For example, we may have an association rule *turkey* $\rightarrow$ *pumpkin pie* along with the calendar pattern *every day in every November*. We call the resulting rules *temporal association rules w.r.t. precise match*.

In some applications, the above temporal association rules may be too restrictive. Instead, we may require that the association rule hold during "enough" number of intervals given by the corresponding calendar pattern. For example, the association rule *turkey* $\rightarrow$ *pumpkin pie* may not hold on every day of every November, but holds on more than 80% of November days. We call such rules *temporal association rules w.r.t. fuzzy match*.

Our data mining problem is to discover from a set of timestamped transactions all temporal association rules w.r.t. precise or fuzzy match for a given calendar schema. We extend an existing algorithm, *Apriori* [2], to discover all such temporal association rules. In addition, based on the observation that the calendar patterns formed from a calendar schema are not isolated but related to each other, we develop two optimization techniques called *temporal apriori-iGen* and *horizontal pruning*, which can be applied to both classes of temporal association rules with some adaptation.

Our contribution in this paper is two-fold. First, we develop a new representation mechanism for temporal association rules based on calendars and identify two classes of interesting temporal association rules. Our representation requires less prior knowledge than the previous methods and the resulting rules are easier to understand. Second, we extend *Apriori* and develop two optimization techniques to discover both classes of temporal association rules. Experiments show that our optimization techniques are effective.

The rest of the paper is organized as follows. The next section defines temporal association rules in terms of calendar patterns. Section 3 extends *Apriori* to discover large itemsets for temporal association rules and presents our optimization techniques. Section 4 presents the experimental evaluation of our algorithms. Section 5 describes the related work, and section 6 concludes the paper.

2 Problem Formulation

2.1 Simple Calendar-based Pattern

A *calendar schema* is a relational schema $R = (G_n : D_n, G_{n-1} : D_{n-1}, \ldots, G_1 : D_1)$ with a *valid* constraint. Each attribute G_i is a granularity name like year, month and week. Each domain D_i is a finite subset of the positive integers. The constraint *valid* is a Boolean function on $D_n \times D_{n-1} \times \cdots \times D_1$, specifying which combinations of the values in $D_n \times D_{n-1} \times \cdots \times D_1$ are "valid". The purpose is to exclude the combinations that we are not interested in or that do not correspond to any time intervals. For example, if we do not want to consider the weekend days and holidays, we can let *valid* evaluate to False for all such days. For brevity, we omit the domains D_i and/or the constraint *valid* from the calendar schema when no confusion arises.

Given a calendar schema $R = (G_n : D_n, G_{n-1} : D_{n-1}, \ldots, G_1 : D_1)$, a *simple calendar-based pattern* (or *calendar pattern* for short) *on* R is a tuple of the form $\langle d_n, d_{n-1}, \ldots, d_1 \rangle$, where each d_i is in D_i or the wild-card symbol $*$. The calendar pattern $\langle d_n, d_{n-1}, \ldots, d_1 \rangle$ represents the set of time intervals intuitively described by "the d_1^{th} G_1 of the d_2^{th} G_2, ..., of d_n^{th} G_n." If d_i is the wild-card symbol '*', then the phrase "the d_i^{th}" is replaced by the phrase "every". For example, given the calendar schema (week, day, hour), the calendar pattern $\langle *, 1, 10 \rangle$ means "the 10th hour on the first day (i.e., Monday) of every week". Each calendar pattern intuitively represents the time intervals given by a set of valid tuples in $D_n \times D_{n-1} \times \cdots \times D_1$.

We say a calendar pattern e *covers* another calendar pattern e' in the same calendar schema if the set of time intervals of e' is a subset of the set of intervals of e. For example, given the calendar schema $(week, day, hour)$, $\langle 1, *, 10 \rangle$ covers $\langle 1, 1, 10 \rangle$. It is easy to see that for a given calendar schema $(G_n, G_{n-1}, \cdots, G_1)$, a calendar pattern $\langle d_n, d_{n-1}, \cdots, d_1 \rangle$ covers another calendar pattern $\langle d'_n, d'_{n-1}, \cdots, d'_1 \rangle$ if and only if for each i, $1 \leq i \leq n$, either d_i = '*' or $d_i = d'_i$.

For simplicity, we require that in a calendar schema $(G_n, G_{n-1}, \ldots, G_1)$, each unit of G_i is contained in a unit of G_{i+1} for $1 \leq i < n$. For example, $(month, day)$ is allowed since each day is contained a unique month. However, the schema $(year, month, week)$ is not allowed because a $week$ may not be contained in a unique $month$.

For the sake of presentation, we call a calendar pattern with k wild-card symbols a *k-star calendar pattern* (denoted e_k), and a calendar pattern with at least one wild-card symbol a *star calendar pattern*. In addition, we call a calendar pattern with no wild-card symbol (i.e., a 0-star calendar pattern) a *basic time interval* if the combination is "valid".

2.2 Temporal Association Rules

Let us first review the concept of association rule, which was originally presented in [1]. Let $\mathcal{I}$ denote a set of data items. Both *transaction* and *itemset* are defined to be subsets of $\mathcal{I}$. Given a set $\mathcal{T}$ of transactions, an *association rule* of the form $X \rightarrow Y$ is a relationship between the two disjoint itemsets X and Y. An association rule satisfies some user-given requirements. The *support* of an itemset by the

set of transactions is the fraction of transactions that contain the itemset. An itemset is said to be *large* if its support exceeds a user-given threshold $minsupport$. The *confidence* of $X \rightarrow Y$ over $\mathcal{T}$ is the fraction of transactions containing X that also contain Y. The association rule $X \rightarrow Y$ *holds* in $\mathcal{T}$ if $X \cup Y$ is large and its confidence exceeds a user-given threshold $minconfidence$.

We assume that each transaction is associated with a *timestamp* (e.g., *November 1, 2000*). Given a basic time interval t (or a calendar pattern e) on a calendar schema, we denote the set of transactions whose timestamps are covered by t (or e) as $\mathcal{T}[t]$ (or $\mathcal{T}[e]$).

Syntactically, a *temporal association rule over a calendar schema* R is a pair (r, e), where r is an association rule and e is a calendar pattern on R. However, multiple meaningful semantics can be associated with temporal association rules. For example, given a set of transactions, one may be interested in the association rules that hold in the transactions on each Monday, or the rules that hold on more than 80% of all Mondays, or the rules that hold in all transactions on all Mondays (i.e., consider the transactions on all Mondays together). In the following, we identify two classes of temporal association rules on which we focus in this paper. Other kinds of temporal association rules may be interesting, but we consider them as possible future work.

Temporal Association Rules w.r.t. Precise Match Given a calendar schema $R = (G_n, G_{n-1}, \cdots, G_1)$ and a set $\mathcal{T}$ of timestamped transactions, a temporal association rule (r, e) *holds w.r.t. precise match* in $\mathcal{T}$ if and only if the association rule r holds in $\mathcal{T}[t]$ for each basic time interval t covered by e. For example, given the calendar schema $(year, month, Thursday)$, we may have a temporal association rule (*turkey* $\rightarrow$ *pumpkin pie*, $\langle *, 11, 4 \rangle$) that holds w.r.t. precise match. This rule means that the association rule *turkey* $\rightarrow$ *pumpkin pie* holds on all Thanksgiving days (i.e., the 4th Thursday in November of every year).

Temporal Association Rules w.r.t. Fuzzy Match Given a calendar schema $R = (G_n, G_{n-1}, \cdots, G_1)$, a set $\mathcal{T}$ of timestamped transactions, and a real number m ($0 < m < 1$, called *match ratio*), a temporal association rule (r, e) *holds w.r.t. fuzzy match* in $\mathcal{T}$ if and only if for at least $100m\%$ of the basic time intervals t covered by e, the association rule r holds in $\mathcal{T}[t]$. For example, given the calendar schema $(year, month, day)$ and the match ratio $m = 0.8$, we may have a temporal association rule (*turkey* $\rightarrow$ *pumpkin pie*, $\langle *, 11, * \rangle$) that holds w.r.t. fuzzy match. This means that the association rule *turkey* $\rightarrow$ *pumpkin pie* holds on at least 80% of the days in November.

Given a calendar schema, we want to discover *all* interesting association rules with *all* their calendar patterns w.r.t precise match and fuzzy match respectively. Note that in many cases, we are not interested in the association rules that only hold during basic time intervals.

3 Finding Large Itemsets

3.1 Overview of Our Algorithms

Mining temporal association rules can be decomposed into two steps: (1) finding all large itemsets for all star calendar patterns on the given calendar schema, and (2) generating temporal association rules using the large itemsets and their calendar patterns. The first step is the crux of the discovery of temporal association rules; in the following, we will focus on this problem. The generation of temporal association rules from large itemsets and their calendar patterns is straightforward and can be resolved using the method discussed in [2].

The algorithm $Apriori$ consists of a number of passes. During pass k, the algorithm tries to find large k-itemsets L_k (i.e., itemsets with k items that have at least the minimum support) from a set of candidates, C_k, through counting the support of each candidate against the entire database. The set of candidates, C_k, is generated from the set of large (k-1)-itemsets, L_{k-1}, ensuring that all $(k-1)$-item subsets of each candidate in C_k are in L_{k-1}.

We extend *Apriori* [2] to discover large itemsets w.r.t. precise and fuzzy match. When precise match is considered, the input of our algorithms consists of a calendar schema R, a set $\mathcal{T}$ of timestamped transactions, and a minimum support $minsupport$. When fuzzy match is considered, an additional input, a match ratio m, is given. Depending on the data mining tasks, our algorithms output the large itemsets for all possible star calendar patterns on R in terms of precise match or fuzzy match.

Figure 1 shows the outline of our algorithms. (This outline is generic for both precise and fuzzy match as well as with and without our optimization techniques discussed later. For different algorithms, appropriate procedures will be supplied.) The algorithms work in passes. In each pass, the basic time intervals in the calendar schema are processed one by one. During the processing of basic time interval e_0 in pass k, the set of large k-itemsets $L_k(e_0)$ is first computed, and then $L_k(e_0)$ is used to update the large k-itemsets for all the calendar patterns that cover e_0.

The first pass is specially handled. In this pass, we compute the large 1-itemsets for each basic time interval by counting the supports of individual items and comparing their supports with $minsupport$. In the subsequent passes, we divide the processing of each basic time interval into three phases. Phase I generates candidate large itemsets for the basic time interval. Phase II reads the transactions whose timestamps are covered by the basic time interval, updates the supports of the candidate large itemsets, and discovers large itemsets for this basic time interval. Phase III uses the discovered large itemsets to update the large itemsets for each star calendar pattern that covers the basic

```
(1) forall basic time intervals e_0 do begin
(2)       L_1(e_0) = {large 1-itemsets in T[e_0]}
(3)       forall star patterns e that cover e_0 do
(4)             update L_1(e) using L_1(e_0);
(5) end
(6) for (k = 2; ∃ a star calendar pattern e such that
        L_{k-1}(e) ≠ ∅; k++) do begin
(7)       forall basic time intervals e_0 do begin
                // Phase I: generate candidates
(8)             generate candidates C_k(e_0);
                // Phase II: scan the transactions
(9)             forall transactions T ∈ T[e_0] do
(10)                  subset (C_k(e_0), T); // c.count++ if
                            //c ∈ C_k(e_0) is contained in T
(11)            L_k(e_0) = {c ∈ C_k(e_0)|c.count ≥
                      minsupport};
                // Phase III: update for star calendar patterns
(12)            forall star patterns e that cover e_0 do
(13)                  update L_k(e) using L_k(e_0);
(14)      end
(15)      Output ⟨L_k(e), e⟩ for all star calendar pattern e.
(16) end
```

Figure 1. Outline of our algorithms for finding large k-itemsets

time interval. At the end of each pass, it outputs the set of large k-itemsets $L_k(e)$ for all star patterns e w.r.t. precise or fuzzy match.

Phase I is the critical step. Indeed, the fewer candidate large itemsets are generated in phase I, the less time phase II will take. Several observations can be used to reduce the number of candidate large itemsets. We will discuss phase I in detail in the following subsections.

Phase II is performed in the same way as in *Apriori* by using the candidate large itemsets generated in phase I. We use a *hash tree* to store all candidate itemsets for a basic time interval e_0 and scan all transactions in $\mathcal{T}[e_0]$ to compute their supports. In Figure 1, function $subset$ traverses the hash tree according to transaction T and increments the supports of the candidate itemsets contained in T. Then the set of large itemsets for e_0 ($L_k(e_0)$) is computed by removing the itemsets that do not have the minimum support.

Now let us explain phase III. After the basic time interval e_0 is processed in pass k, the large k-itemsets for all the calendar patterns that cover e_0 are updated as follows. For precise match, this is done by intersecting the set $L_k(e_0)$ of large k-itemset for the basic time interval e_0 with the set $L_k(e)$ of large k-itemsets for the calendar pattern e (i.e., $L_k(e) = L_k(e) \cap L_k(e_0)$). (Certainly, $L_k(e) = L_k(e_0)$ when $L_k(e)$ is updated for the first time.) It is easy to see that after all the basic time intervals are processed, the set of large k-itemsets for each calendar pattern consists of the k-itemsets that are large for all basic time intervals covered by the pattern.

Update for fuzzy match is a little more complex. We associate a counter *c_update* with each candidate large itemset for each star calendar pattern. The counters are initially set to 1. When $L_k(e_0)$ is used to update $L_k(e)$ in phase III, the counters of the itemsets in $L_k(e)$ that are also in $L_k(e_0)$ are incremented by 1, and the itemsets that are in $L_k(e_0)$ but not in $L_k(e)$ are added to $L_k(e)$ with the counter set to 1. Suppose there are totally N basic time intervals covered by e and this is the n-th update to $L_k(e)$. We drop an itemset if its counter c_update does not satisfy $c\_update + (N - n) \geq m \cdot N$. It is easy to see that a dropped itemset cannot be large for e; on the other hand, if an itemset remains in $L_k(e)$, then its counter $c\_update \geq m \cdot N$ since it is not dropped in the last update.

3.2 Generating Candidate Large Itemsets

3.2.1 Direct-Apriori

A naive approach to generating candidate large itemsets is to treat each basic time interval individually and directly apply *Apriori*'s method for candidate generation. We call this approach *Direct-Apriori* (for precise or fuzzy match depending on the context). Phase I of *Direct-Apriori* is instantiated as follows.

$$C_k(e_0) = aprioriGen(L_{k-1}(e_0))$$

Here function $aprioriGen$ is used to generate $C_k(e_0)$, the set of candidate large k-itemsets, from the set of large (k-1)-itemsets, $L_{k-1}(e_0)$, ensuring that all $(k-1)$-item subsets of each candidate in $C_k(e_0)$ are in $L_{k-1}(e_0)$.

According to *Apriori* [2], the set of candidate large k-itemsets, $C_k(e_0)$, is a super set of all the large k-itemsets for e_0. Thus, phase II of the algorithm will correctly generate the set of large k-itemsets for e_0. By the argument in subsection 3.1, for each calendar star pattern e, $L_k(e)$ will consist of the k-itemsets that are large (w.r.t. precise or fuzzy match) for e after all the basic time intervals are processed.

3.2.2 Temporal-Apriori

Note that in many cases, we may not be interested in the association rules that only hold in basic time intervals. Direct-Apriori does not take this into consideration and thus may lead to unnecessary data processing. In the following, we present two optimization techniques, which we call *temporal aprioriGen* and *horizontal pruning*, to improve the candidate generation for situations where temporal association rules for basic time intervals are not considered. The resulting algorithm is called *Temporal-Apriori* (for precise or fuzzy match according to the context).

Temporal aprioriGen is partially based on the assumption mentioned above. Since we do not need to find the large itemsets for basic time intervals, we do not need to count the supports for all the potentially large k-itemsets generated by $C_k(e_0) = aprioriGen(L_{k-1}(e_0))$ for each basic time interval e_0. Indeed, we only need the supports of the itemsets that are potentially large for some star calendar patterns that covers e_0. In other words, given a basic time interval e_0, if a candidate large k-itemset cannot be large for any of the star calendar patterns that cover e_0, we can ignore it even if it could be large for e_0. Therefore, we can generates candidates $C_k(e_0)$ as follows.

$$C_k(e_0) = \bigcup_{e \ covers \ e_0} aprioriGen(L_{k-1}(e) \cap L_{k-1}(e_0))$$

Example 1 Consider a fuzzy-match temporal association rule discovery using the calendar schema $R = (\textit{week} : \{1, \cdots, 5\}, \textit{day} : \{1, \cdots, 7\})$. Suppose we have computed the following large 2-itemsets: $L_2(\langle 3, 2\rangle) = \{AB, AC, AD, AE, BD, CD, CE\}$, $L_2(\langle *, 2\rangle) = \{AB, AC, AD, BC, BD, CE\}$, $L_2(\langle 3, *\rangle) = \{AB, AC, AD, BD, CD\}$, and $L_2(\langle *, *\rangle) = \{AB, AD, BD, CD, AC, AE\}$. To compute the candidate large 3-itemsets for the basic time interval $\langle 3, 2\rangle$, we first get $L_T = L_2(\langle *, 2\rangle) \cap L_2(\langle 3, 2\rangle) = \{AB, AC, AD, BD, CE\}$ and then generate $C_3(\langle *, 2\rangle) = \textit{aprioriGen}(L_T) = \{ABD\}$. Similarly, we can get $C_3(\langle 3, *\rangle) = \{ABD, ACD\}$ and $C_3(\langle *, *\rangle) = \{ABD, ACE\}$. Then the set of candidate large 3-itemsets is $C_3(\langle 3, 2\rangle) = C_3(\langle *, 2\rangle) \cup C_3(\langle 3, *\rangle) \cup C_3(\langle *, *\rangle) = \{ABD, ACD, ACE\}$. □

The above method works well for both precise and fuzzy match. (Note that we can use $L_{k-1}(e)$ instead of $L_{k-1}(e) \cap L_{k-1}(e_0)$ for precise match, since $L_{k-1}(e)$ is a subset of $L_{k-1}(e_0)$.) Moreover, we can improve the candidate generation for precise match on the basis of the following Lemma.

Lemma 1 *Given a star calendar pattern e, an itemset is large for e w.r.t. precise match only if it is large w.r.t. precise match for all 1-star calendar patterns covered by e.*

Consider $C_k(e_0)$, the set of candidate large k-itemset for e_0. We only need $C_k(e_0)$ to generate large itemsets for patterns e that cover e_0. According to Lemma 1, an itemset is large for a given star calendar pattern e only if it is large for all 1-star calendar patterns covered by e. Thus, we can generate the candidate large k-itemsets ($k > 1$) for precise match as follows.

$$C_k(e_0) = \bigcup_{e_1 \ covers \ e_0} aprioriGen(L_{k-1}(e_1))$$

Example 2 Consider a precise-match temporal association rule discovery using the calendar schema $R = (\textit{week} : \{1, \cdots, 5\}, \textit{day} : \{1, \cdots, 7\})$. Suppose we have the following large 2-itemsets: $L_2(\langle 3, 2\rangle) = \{AB, AC, AD, AE, BC, BD, CD, CE\}$, $L_2(\langle *, 2\rangle) = \{AB, AC, AD, BC, BD, CE\}$, and $L_2(\langle 3, *\rangle) = \{AB, AC, AD, BD, CD\}$. To compute the candidate large 3-itemsets for $\langle 3, 2\rangle$, we will first generate $C_3(\langle *, 2\rangle) = \{ABC, ABD\}$ and $C_3(\langle 3, *\rangle) = \{ABD, ACD\}$. Then the set of candidate large 3-itemsets is $C_3(\langle 3, 2\rangle) = C_3(\langle *, 2\rangle) \cup C_3(\langle 3, *\rangle) = \{ABC, ABD, ACD\}$. In contrast, if we use Direct-Apriori, we will generate the candidates from $L_2(\langle 3, 2\rangle)$ and have the set of candidate large 3-itemsets as $C'_3(\langle 3, 2\rangle) = \{ABC, ABD, ACD, ACE, BCD\}$. □

Our second optimization technique, *horizontal pruning*, is also based on Lemma 1, but applied during a pass. We first discuss the precise match case. Consider pass k. For each basic time interval e_0, we update (among others) $L_k(e_1)$ for each e_1 that covers e_0. After the first time $L_k(e_1)$ is updated, for every e_0 processed, we update $L_k(e_1)$ to be $L_k(e_1) \cap L_k(e_0)$, i.e., drop the itemsets in $L_k(e_1)$ that do not appear in $L_k(e_0)$. Hence, after the first time $L_k(e_1)$ is updated, $L_k(e_1)$ always contains all the large k-itemsets for e_1 (plus other itemsets that will eventually be dropped). In other words, at any time of the processing (except before the first update), if a k-itemset l does not appear in $L_k(e_1)$, then l is not large for e_1.

Now we can use the tentative $L_k(e_1)$ (i.e., updated at least once) to prune the candidate large k-itemsets in $C_k(e_0)$ as follows. If an itemset l in $C_k(e_0)$ does not appear in any of the tentative $L_k(e_1)$, where e_1 is a 1-star pattern that covers e_0, then l cannot be large for any star pattern e that covers e_0. Indeed, any star pattern e covering e_0 must cover at least one of the 1-star patterns that cover e_0. Let this particular 1-star pattern be e'_1. Since l is not large for any 1-star pattern that covers e_0, l is not large for e'_1. By Lemma 1, l cannot be large for e, and we may drop l from $C_k(e_0)$.

Example 3 Let us continue example 2. Suppose when $\langle 3, 2\rangle$ is being processed, we already have $L_3(\langle *, 2\rangle) = \{ABD\}$ and $L_3(\langle 3, *\rangle) = \{ABD, ACD\}$. Then we can further prune $C_3(\langle 3, 2\rangle)$, which is $\{ABC, ABD, ACD\}$, by $C_3(\langle 3, 2\rangle) = C_3(\langle 3, 2\rangle) \cap (L_3(\langle *, 2\rangle) \cup L_3(\langle 3, *\rangle)) = \{ABD, ACD\}$. □

Horizontal pruning for precise match cannot be directly applied to fuzzy match. This is because fuzzy match allows a large itemset to be small for *some* basic time intervals. Nevertheless, a similar idea can be applied to fuzzy match. The idea is based on the observation that an itemset is not large for a calendar pattern if it is not large for a certain number of basic time intervals covered by the pattern. For example, an itemset l can never be large for 80% of all Mondays if it is already known not to be large for 20% of the Mondays. Therefore, we discard the candidate large itemsets (for e_0) that cannot be large for any star pattern e that covers e_0 even if these itemsets are large for e_0.

Example 4 Let us continue example 1. We already have $C_3(\langle 3, 2\rangle) = \{ABD, ACD, ACE\}$. Suppose all of $L_3(\langle *, 2\rangle)$, $L_3(\langle 3, *\rangle)$, and $L_3(\langle *, *\rangle)$ have been up-

dated at least once. Then we can update a copy of $L_3(\langle *, 2\rangle)$ with $C_3(\langle 3, 2\rangle)$ and get the result, for example, $C_3(\langle *, 2\rangle) = \{ABD, ABE\}$. If we also get $C_3(\langle 3, *\rangle) = \{ABD, ACD\}$ and $C_3(\langle *, *\rangle) = \{ABD\}$, then $C_3(\langle 3, 2\rangle)$ can be pruned as $C_3(\langle 3, 2\rangle) = C_3(\langle 3, 2\rangle) \cap (C_3(\langle *, 2\rangle) \cup C_3(\langle 3, *\rangle) \cup C_3(\langle *, *\rangle)) = \{ABD, ACD\}$. □

We prove the correctness of Temporal-Apriori for precise match as follows. First, we show that the algorithm has the same output as Direct-Apriori if for each e_0, it uses a super set of the union of large k-itemsets for all 1-star calendar patterns that cover e_0. Then we prove the equivalence of Temporal-Apriori and Direct-Apriori by showing that the set of candidate large k-itemsets used for each basic time interval in Temporal-Apriori is such a super set. The correctness of Temporal-Apriori for fuzzy match is proved similarly. This result is summarized in the following Lemmas and Theorems. Please refer to [8] for the details.

Lemma 2 *If Temporal-Apriori for precise match uses a super set of $\bigcup_{e_1\ covers\ e_0} L_k(e_1)$ as the set of candidate large k-itemsets for each e_0, then it has the same output as Direct-Apriori for precise match.*

Theorem 1 *Temporal-Apriori for precise match is equivalent to Direct-Apriori for precise match.*

Lemma 3 *If Temporal-Apriori for fuzzy match uses a super set of $\bigcup_{e\ covers\ e_0} L_k(e)$ as the set of candidate large k-itemsets for each e_0, then it has the same output as Direct-Apriori for fuzzy match.*

Theorem 2 *Temporal-Apriori for fuzzy match is equivalent to Direct-Apriori for fuzzy match.*

4 Experiments

To evaluate the performance of our algorithms and optimization techniques, we performed a series of experiments using both a real-world data set (KDD Cup 2000 [7]) and synthetic data sets. Due to space reasons, we only report the results on the synthetic data sets in this paper. A detailed description of the experiments is available in [8].

In order to generate data sets with various characteristics, we extend the data generator proposed in [2] to incorporate temporal features. For each basic time interval e_0 in a given calendar schema, we first generate a set of maximal potentially large itemsets called *per-interval itemsets* and then generate transactions $\mathcal{T}[e_0]$ from per-interval itemsets following the exact method in [2]. Specifically, the sizes of each transaction and each per-interval itemset are picked from Poisson distribution with mean μ equal to $|T|$ and $|I|$ respectively. Each per-interval itemset is generated by copying a half of its items from its previous one (the first per-interval itemset is generated totally randomly), and randomizing the other half. Each per-interval itemset is also assigned a weight from an exponential distribution with unit mean. After all per-interval itemsets are generated, each transaction is formed by incorporating a set of per-interval itemsets selected according to their weights.

Notation	Meaning	Default value
$\|D\|$	Number of transactions per basic time interval	10,000
$\|T\|$	Avg. size of the transactions	10
$\|I\|$	Average size of the maximal potentially large itemsets	4
$\|L\|$	Num. of per-interval itemsets	1,000
N	Num. of items	1,000
P_r	Pattern-ratio	0.4
N_p	Num. of star calendar patterns per pattern itemset	40

Figure 2. Parameters for data generation

To model the phenomenon that some itemsets may have temporal patterns but others may not, we choose a subset of the per-interval itemsets from a common set of itemsets called *pattern itemsets* shared by all basic time intervals but generate the others independently for each basic time interval. We use a parameter *pattern-ratio*, denoted P_r, to decide the percentage of per-interval itemsets that should be chosen from the pattern itemsets.

To decide which pattern itemsets should be used for a basic time interval, we associate several star calendar patterns with each pattern itemset. For each basic time interval, we choose itemsets repeatedly and randomly from the pattern itemsets until we have enough number of pattern itemsets. Each time when a pattern itemset is chosen, we use it as a per-interval itemset if it has a calendar pattern that covers the basic time interval; otherwise, the itemset is ignored. We use a parameter N_p to adjust this feature such that the number of calendar patterns assigned to each pattern itemset conforms to a Poisson distribution with mean N_p.

The calendar patterns assigned to pattern itemsets are selected from the space of all star calendar patterns. In order to model the phenomenon that the calendar patterns covering more basic time intervals are less possible than those covering fewer ones, we associate with each calendar pattern a weight, which corresponds to the probability that this calendar pattern is selected. The weight of a calendar pattern is set to 0.5^k, where k is the number of wild-card symbols in the calendar pattern. The weight is then normalized so that the sum of the weights of all calendar patterns is 1.

Our data generation procedure takes the calendar schema $(year : \{1995 - 1999\}, month, day)$ and the parameters shown in Figure 2. To examine the performance of the algorithms, we generated a series of data sets by varying one parameter while keeping others at their default values. The size of the data sets ranges from 739 MB to 5.41 GB.

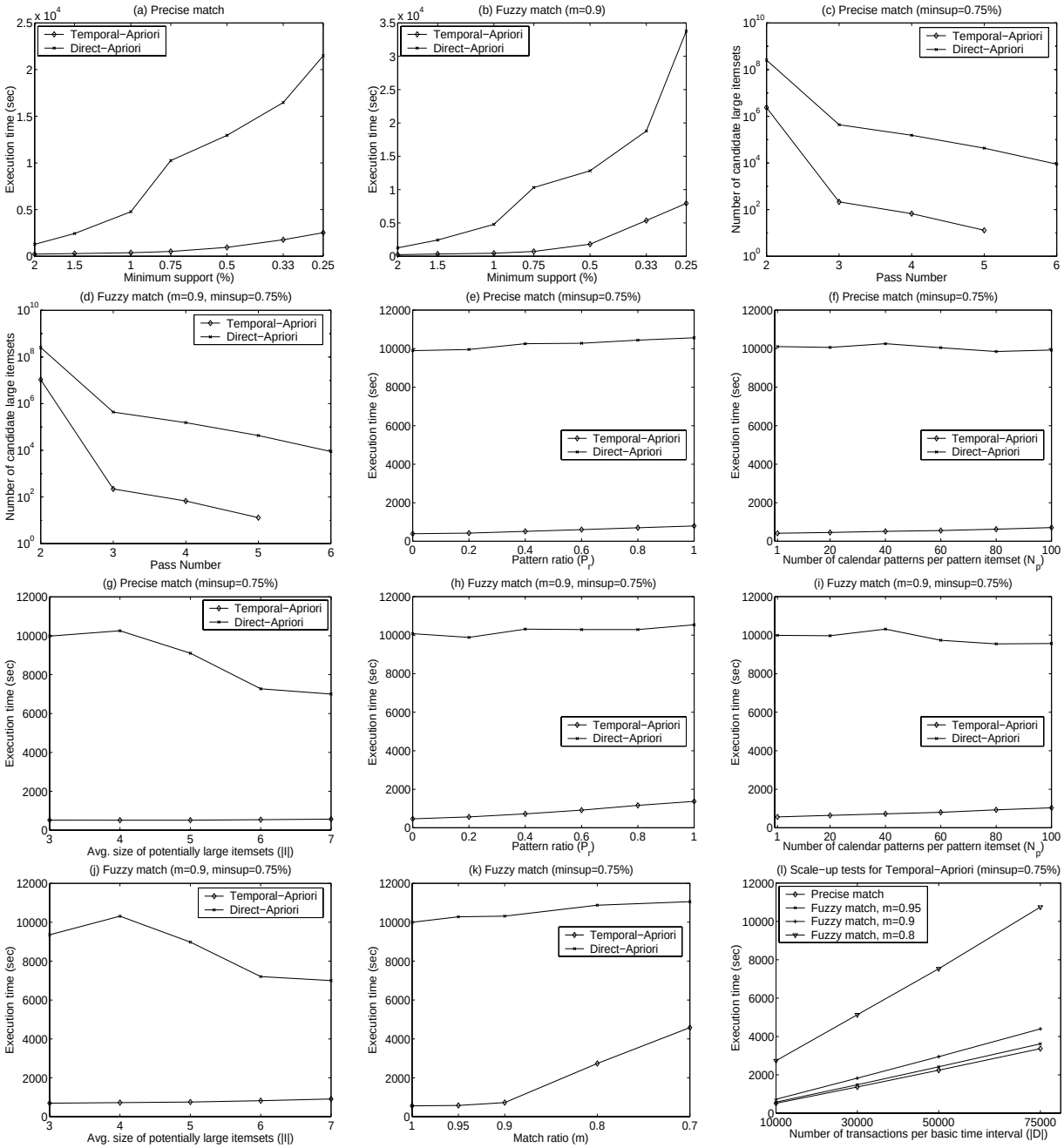

Figure 3. Experimental result on synthetic data sets

Figures 3(a) and 3(b) show the effectiveness of our optimization techniques. For precise match, Temporal-Apriori is 5 to 22 times faster than Direct-Apriori; for fuzzy match, Temporal-Apriori is 2.5 to 12 times faster than Direct-Apriori. Figures 3(c) and 3(d) give the total number of candidate large itemsets for the experiments with the minimum support 0.75%, showing that the optimization techniques greatly reduced the number of candidates in each pass. In figures 3(e) through (k), we generate different data sets by varying parameter P_r, N_p and m. In all experiments, Temporal-Apriori performs significantly better than Direct-Apriori. Figure 3(l) shows that Temporal-Apriori scales well when the number of transactions grows large.

5 Related Work

Since the concept of association rule was first introduced in [1], discovery of association rules has been extensively studied. The concept of association rule was also extended in several ways, including generalized rules and multi-level rules (e.g., [6]), quantitative rules (e.g., [12]), and constraint-based rules (e.g., [4]). Among these extensions is the discovery of temporal association rules.

There are several kinds of meaningful temporal association rules [3, 5, 10, 9, 11]. The problem of mining cyclic association rules (i.e., the association rules that occur periodically over time) has been studied in [9]. Several algorithms and optimization techniques were presented and shown effective; however, this work is limited in that it cannot deal with multiple granularities and cannot describe real-life concepts such as *the first business day of every month*. In [11], the work in [9] was extended to approximately discover user-defined temporal patterns in association rules. The work in [11] is more flexible and practical than [9]; however, it requires user-defined calendar algebraic expressions in order to discover temporal patterns. Indeed, this is to require user's prior knowledge about the temporal patterns to be discovered.

Our work differs from [9] and [11] in that instead of using cyclic patterns or user-defined calendar algebraic expressions, we use calendar schema as a framework for temporal patterns. As a result, our approach usually requires less priori knowledge than [9] and [11] (i.e. we need not know or describe each individual "interesting" temporal pattern). In addition, unlike [11], which discover temporal association rules for one user-defined temporal pattern, our approach considers all possible temporal patterns in the calendar schema, thus we can potentially discover more temporal association rules.

6 Conclusion and Future Work

In this paper, we proposed two classes of temporal association rules, *temporal association rules w.r.t. precise match* and *temporal association rules w.r.t. fuzzy match*, to represent regular association rules along with their temporal patterns in terms of calendar schemas. An immediate advantage is that the corresponding data mining problem requires less prior knowledge than the prior methods and hence may discover more unexpected rules. In addition, we extended *Apriori* to discover temporal association rules w.r.t. both precise match and fuzzy match. To deal with the situation when we are not interested in the temporal association rules involving only basic time intervals (i.e., time intervals represented by a calendar pattern with 0 wildcard symbols), we developed two optimization techniques by studying the relationships among calendar patterns. Our experiments showed that our optimization techniques are quite effective. Similar optimization techniques apply to situations where we are only interested in time intervals represented by a calendar pattern with at least $k > 1$ wildcard symbols.

The future work includes two directions. First, we would like to explore other meaningful semantics of temporal association rules and extend our techniques to solve the corresponding data mining problems. Second, we would like to consider temporal patterns in other data mining problems such as clustering.

References

[1] R. Agrawal, T. Imielinski, and A. N. Swami. Mining association rules between sets of items in large databases. In *Proc. of the 1993 Int'l Conf. on Management of Data*, pages 207–216, 1993.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules in large databases. In *Proc. of the 1994 Int'l Conf. on Very Large Data Bases*, pages 487–499, 1994.

[3] J. Ale and G. Rossi. An approach to discovering temporal association rules. In *Proc. of the 2000 ACM Symposium on Applied Computing*, pages 294–300, 2000.

[4] R. Bayardo Jr., R. Agrawal, and D. Gunopulos. Constraint-based rule mining in large, dense databases. In *Proc. of the 15th Int'l Conf. on Data Engineering*, pages 188–197, 1999.

[5] X. Chen and I. Petrounias. Mining temporal features in association rules. In *Proc. of the 3rd European Conf. on Principles and Practice on Knowledge Discovery in Databases*, pages 295–300, 1999.

[6] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. of 21th Int'l Conf. on Very Large Data Bases*, pages 420–431, 1995.

[7] R. Kohavi and C. Brodley. 2000 knowledge discovery and data mining cup. Data for the Cup was provided by Blue Martini Software and Gazelle.com, 2000. http://www.ecn.purdue.edu/KDDCUP/.

[8] Y. Li, P. Ning, X. S. Wang, and S. Jajodia. Discovering calendar-based temporal association rules. Manuscript. http://www.ise.gmu.edu/∼pning/tdm.ps, Nov. 2000.

[9] B. Özden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proc. of the 14th Int'l Conf. on Data Engineering*, pages 412–421, 1998.

[10] C. Rainsford and J. Roddick. Adding temporal semantics to association rules. In *Proc. of the 3rd European conf. on principles and practice of knowledge discovery in databases*, pages 504–509, 1999.

[11] S. Ramaswamy, S. Mahajan, and A. Silberschatz. On the discovery of interesting patterns in association rules. In *Proc. of the 1998 Int'l Conf. on Very Large Data Bases*, pages 368–379, 1998.

[12] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. In *Proc. of the 1996 ACM SIGMOD Int'l Conf. on Management of Data*, pages 1–12, 1996.