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

{*yli, pning, xywang, jajodia*}*@ise.gmu.edu*

Abstract

A temporal association rule is an association rule that holds during specific time intervals. An example can be that eggs and coffee are frequently sold together in morning hours. This paper studies temporal association rules during 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 consisting of all the 16th days of all months 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. Compared to the non-temporal association rule discovery, temporal association rules are more difficult to find due to the usually large number of possible temporal patterns for a given calendar schema. 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 paper then studies the performance of the algorithms by using a real-world data set as well as synthetic data sets. The performance data show that the algorithms and related optimization techniques are effective.

1 Introduction

Among various types of data mining applications, the analysis of transactional data has been considered important. It is assumed that the database keeps information about user transactions, where each transaction is a collection of data items. The notion of *association rule* was proposed to capture the cooccurrence of items in transactions [AIS93]. For example, given a database of orders (transactions) placed in a restaurant, we may have an association rule of the form

egg→ *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 above are commonly referred to as *support* and *confidence* respectively.

One interesting extension to association rules is to include a temporal dimension. For example, *eggs* and *coffee* may be ordered together primarily between 7AM and 11AM. Therefore, we may find that the above association rule has a support as high as 40% among the transactions that happened between 7AM and 11AM and has a support as low as 0.005% in other transactions. As another 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*. That is, the association rule "*turkey* → *pumpkin pie*" has a high support and a high confidence in the transactions that happened 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.

Informally, we refer to the association rules along with their temporal intervals as *temporal association rules*. The discovery of temporal association rules has been discussed in the literature. For example, in [ÖRS98], discovery of cyclic association rules (i.e., the association rules that occur periodically over time) was studied. However, periodicity has limited expressiveness in describing real-life concepts such as *the first business day of every month* since the distances between two consecutive such business days are not constant. In general, the model does not deal with calendric concepts like year, month, day, etc. In [RMS98], the work in [ÖRS98] was extended to treat user-defined temporal patterns. Although the work in [RMS98] is more flexible than that of [ÖRS98], it only considers the association rules that hold during the time intervals described by a user-given calendar algebraic expression. In other words, a single set of time intervals is given by the user and only the association rules on these intervals are considered. This method hence requires user's prior knowledge about the temporal patterns. (Other related work will be discussed later in the paper.)

In this paper, we propose an approach to discovering temporal association rules with relaxed requirement of prior knowledge. Instead of using cyclic or user-defined calendar algebraic expressions, we use calendar schemas as frameworks for discovering temporal patterns. Our approach is not a simple extension of the previous approaches, because we assume less prior knowledge and, therefore, we have more potential temporal patterns to explore. It is necessary to explore optimization opportunities afforded by the relationships among the temporal patterns in order to achieve the performance and scalability for practical uses.

A calendar schema is determined by a hierarchy of calendar concepts. 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. Based on the work for generating user-defined calendars, e.g., [LMF86], cyclic patterns of [ÖRS98] and calendar algebra expressions of [RMS98] can be considered as special cases of calendar patterns.

We assume that the transactions are timestamped so we can decide if a transaction happens during a specific time interval. Given a set of transactions and a calendar schema, our first interest is to discover all the association rule and calendar pattern pairs 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*, to discover all such temporal association rules. In addition, we observe that the calendar patterns formed from a calendar schema are not isolated but related to each other. We use the relationship to develop two optimization techniques called *temporal aprioriGen* and *horizontal pruning*. These optimization techniques 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 on the basis of calendars and identify two classes of interesting temporal association rules: temporal association rules w.r.t. precise match and temporal association rules w.r.t. fuzzy match. Our representation requires less prior knowledge than the prior methods and the resulting time intervals are easier to understand. Second, we extend the algorithm *Apriori* and develop two optimization techniques to discover both classes of temporal association rules. Our experiments demonstrate that our optimization techniques are effective.

The rest of the paper is organized as follows. In section 2, we define temporal association rules in terms of calendar patterns in the framework of calendar schemas. In section 3, we extend *Apriori* to discover large itemsets for temporal association rules and present our optimization techniques to improve the performance and scalability. In section 4, we present the experimental evaluation of our algorithms using both real and synthetic data sets. Section 5 presents the related work, and section 6 concludes the paper with some discussions. Appendix A gives the proof of the lemmas and theorems that appear in the paper.

2 Problem Formulation

2.1 Association Rule

The concept of association rules, which was motivated by market basket analysis and originally presented in [AIS93], has been discussed in many application domains. Let $\mathcal{I}$ 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 $\mathcal{I}$. 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 $minconfidence$. (There are constraints other than user-specified minimum support and minimum confidence that define interesting rules, e.g., minimum improvement constraint [BAG99]. However, they are out of the scope of this paper.)

2.2 Simple Calendar-based Pattern

In the following, we present a class of calendar related temporal patterns called *simple calendar-based patterns*.

A *calendar schema* is a relational schema (in the sense of relational databases) $R = (f_n : D_n, f_{n-1} : D_{n-1}, \ldots, f_1 : D_1)$ together with a *valid* constraint (explained below). Each attribute f_i is a calendar unit name like year, month, and week etc. Eeach 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 $D_n \times \cdots \times D_1$ are "valid". This constraint serves two purposes. The first is to exclude the combinations that do not correspond to any time intervals due to the interaction of the calendar units. For example, we may have a calendar schema $(year : \{1995, 1996, \cdots, 1999\}, month : \{1, 2, \cdots, 12\}, day : \{1, 2, \cdots, 31\})$ with the constraint *valid* that evaluates $\langle y, m, d \rangle$ to True only if the combination gives a valid date[1]. The second purpose of the *valid* constraint is to exclude the time intervals that we are not interested in. 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 may omit the domains D_i and/or the constraint *valid* from the calendar schema when no confusion arises.

Given a calendar schema $R = (f_n : D_n, f_{n-1} : D_{n-1}, \ldots, f_1 : D_1)$, a *simple calendar-based pattern* (or *calendar pattern* for short) *on the calendar schema* R is a tuple on R 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 $*$. Here, we choose to use d_i for both an element of D_i and the symbol $*$ to simplify our notation. The calendar pattern $\langle d_n, d_{n-1}, \ldots, d_1 \rangle$ represents the set of time intervals that are intuitively described by "the d_1^{th}

[1] For example, $\langle 1995, 1, 3 \rangle$ is a valid date while $\langle 1996, 2, 31 \rangle$ is not.

f_1 of the d_2^{th} f_2, ..., of d_n^{th} f_n." In the above description, if d_i is the wild-card symbol '*' (instead of an integer), 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 weeks". Similarly, $\langle 1, *, 10 \rangle$ represents the time intervals "the 10th hour of every day of week 1". Each calendar pattern in effect represents the time intervals given by a set of tuples in $D_n \times \cdots \times D_1$ that are valid. For simplicity, we omit a more formal treatment of the above concepts.

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 $(week, day, hour)$, $\langle 1, *, 10 \rangle$ (i.e., the 10th hour of every day of week 1) covers $\langle 1, 1, 10 \rangle$ (i.e., the 10th hour of day 1 of week 1). It is easy to see that for a given calendar schema $(f_n, f_{n-1}, \cdots, f_1)$, 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 $d_i = d'_i$.

Simple calendar-based patterns give a simple and intuitive representation of sets of time intervals in terms of a calendar schema. Note that time intervals or periodic cycles can be easily described by calendar patterns with appropriate calendar schemas having perhaps user-defined calendars. For example, the periodic cycle "every seventh day" can be expressed by a calendar pattern $\langle *, i \rangle$, where $1 \leq i \leq 7$, under the calendar schema $R = (week, day)$ depending on which day the cycle starts.

For simplicity, we require that in a calendar schema $(f_n, f_{n-1}, \ldots, f_1)$, each calendar unit of f_i is uniquely contained in a unit of f_{i+1}, for $1 \leq i < n$. For example, $(month, day)$ is allowed since each day is covered by a unique month. However, the schema $(year, month, week)$ is not allowed because a $week$ may not be contained in a unique $month$. It is often convenient and sometimes necessary for users to define calendar units and then use them in calendar schemas. For example, the 24 hours of a day may be partitioned into five parts, representing *early morning, morning, work hour, night,* and *late night* respectively, forming a new calendar unit. The reader is referred to [LMF86] and [BJW00] for generation of user-defined calendars.

For brevity of presentation, we introduce some notations for calendar patterns. We call a calendar pattern with exactly 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* under the calendar schema if the combination is valid with respect to the constraint given in the calendar schema. In other words, a basic time interval corresponds to a tuple in $D_n \times \cdots \times D_1$ that is valid.

2.3 Temporal Association Rules

We assume that each transaction is associated with a *timestamp* that gives the time of the transaction. For example, a transaction may be associated with a timestamp that represents *November 1, 2000*, which indicates that the transaction occurred on *November 1, 2000*. Given a basic time interval t (or a calendar pattern e) under a given 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 will 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 $R = (f_n, f_{n-1}, \cdots, f_1)$ 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* → *pumpkin pie*, $\langle *, 11, 4\rangle$) that holds w.r.t. precise match. This rule means that the association rule *turkey* → *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 $R = (f_n, f_{n-1}, \cdots, f_1)$, 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* → *pumpkin pie*, $\langle *, 11, *\rangle$) that holds w.r.t. fuzzy match. This means that the association rule *turkey* → *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 in the given calendar schema. Specifically, we attack the following two data mining problems:

1. (Precise match) Given a calendar schema R and a set $\mathcal{T}$ of timestamped transactions, find all temporal association rules (r, e) that hold w.r.t. precise match in $\mathcal{T}$.

2. (Fuzzy match) Given a calendar schema R, a set $\mathcal{T}$ of timestamped transactions, and a match ratio m, find all temporal association rules (r, e) that hold w.r.t. fuzzy match in $\mathcal{T}$.

We further assume that we are not interested in the association rules that only hold during basic time intervals. Indeed, such rules do not reveal much information in terms of time. Therefore, we exclude the 0-star calendar patterns e_0 from the output of our data mining problems.

Now we introduce some additional notations for the sake of presentation. For a basic time interval t in a calendar schema, we say an itemset is *large for t* if it is large in $\mathcal{T}[t]$. For a calendar pattern e, we say an itemset is *large for e w.r.t. precise match* if it is large for each basic time interval covered by e. Further consider a fuzzy match ratio m, we

```
(1) $L_1$ ={large 1-itemsets};
(2) for (k=2; $L_{k-1} \neq \emptyset$; $k++$) do begin
(3)     $C_k$ = aprioriGen($L_{k-1}$); // New candidates
(4)     forall transactions $T \in \mathcal{T}$ do
(5)         subset($C_k, T$); // $c.count++$ if $c \in C_k$ is contained in $T$
(6)     $L_k = \{c \in C_k | c.count \geq minsupport\}$;
(7) end
(8) Answer = $\bigcup_k L_k$;
```

Figure 1: Algorithm $Apriori$.

say an itemset is *large for* e *w.r.t. fuzzy match* if it is large for at least $100m\%$ of the basic time intervals covered by e.

3 Finding Large Itemsets

Mining temporal association rules can be decomposed into two subproblems: (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. Finding large itemsets along with their calendar patterns 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 [AS94].

Our approaches to finding large itemsets for all calendar patterns are based on *Apriori* [AS94]. Before going into details of our approaches, we briefly go over Apriori.

3.1 *Apriori*

Figure 1 shows the outline of *Apriori*. The algorithm $Apriori$ consists of a number of passes. During pass k, the algorithm tries to find large k-itemsets (i.e., itemsets with k items that have at least the minimum support). It first generates C_k, the set of candidate large k-itemsets, then counts the support of each candidate itemset by scanning all the transactions in the data set, and finally finds L_k, the set of large k-itemsets, by inspecting the supports of all the candidate itemsets. The algorithm terminates when no large itemset is discovered after a pass.

Function $aprioriGen$ is a critical step of *Apriori* (step 3). It constructs the set of candidate large k-itemsets, C_k, 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}. It turns out that *aprioriGen* is very effective in reducing the size of the candidate set [AS94].

Scanning the transactions and updating the supports of candidate itemsets (step 4 and 5) are the most time-

```
forall basic time intervals $e_0$ do begin
    $L_1(e_0) = \{$large 1-itemsets in $\mathcal{T}[e_0]\}$
    forall star patterns $e$ that cover $e_0$ do
        update $L_1(e)$ using $L_1(e_0)$;
end
for ($k = 2$; $\exists$ a star calendar pattern $e$ such that $L_{k-1}(e) \neq \emptyset$; $k++$) do begin
    forall basic time intervals $e_0$ do begin
        // Phase I: generate candidates
        generate candidates $C_k(e_0)$;
        // Phase II: scan the transactions
        forall transactions $T \in \mathcal{T}[e_0]$ do
            subset ($C_k(e_0), T$); // $c.count++$ if $c \in C_k(e_0)$ is contained in $T$
        $L_k(e_0) = \{c \in C_k(e_0) | c.count \geq minsupport\}$;
        // Phase III: update for star calendar patterns
        forall star patterns $e$ that cover $e_0$ do
            update $L_k(e)$ using $L_k(e_0)$;
    end
    Output $\langle L_k(e), e \rangle$ for all star calendar pattern $e$.
end
```

Figure 2: Outline of our algorithms for finding large k-itemsets

consuming steps, since they require access to both disk and a potentially large set of candidate itemsets. *Apriori* uses a *hash tree* to store all candidate itemsets and their supports [AS94]. In Figure 1, function $subset$ traverses the hash tree according to the transaction T and increments the supports of the candidate itemsets contained in T.

3.2 Overview of Our Algorithms

We extend *Apriori* 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 2 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 subroutines will be supplied.) As Apriori, 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[2], and then $L_k(e_0)$ is used to update the large k-itemsets for all the calendar patterns that cover e_0. Note that although our data mining tasks do not need association rules for basic time intervals, the large itemsets for basic time intervals $L_k(e_0)$ are used in the algorithms for efficiency considerations. Indeed, assume we have the calendar schema $($year, month, day$)$. In the algorithms, we may need to consider, e.g., the calendar patterns ⟨1995, *, 1⟩ as well as ⟨*,1,*⟩. These two patterns have an overlapping basic time interval, namely ⟨1995, 1, 1⟩. In our algorithms, we use the large itemsets for ⟨1995, 1, 1⟩ (and for other basic time intervals) to derive the large itemsets for ⟨1995, *, 1⟩ and ⟨*,1,*⟩ to avoid duplicate tasks. This strategy is reflected in lines (4) and (13).

The first pass is specially handled. In the first 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 from the previously generated large itemsets. 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 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.

Similar to the discovery of non-temporal association rules, phase I is the critical step in mining temporal association rules. Indeed, the fewer candidate large itemsets are generated, the less time phase II will take, and the better performance can be achieved. 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.

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 covers 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, when $L_k(e)$ is updated for the first time, we let $L_k(e) = L_k(e_0)$.) 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 $L_k(e)$. It is easy to see that an itemset cannot be large for e if its counter *c_update*

[2] When some of our optimization techniques are used, a subset of the large k-itemsets for e_0 may be used as $L_k(e_0)$ as explained later.

$L_2(\langle *, 2\rangle)$ (Before update)	$L_2(\langle 3, 2\rangle)$	$L_2(\langle *, 2\rangle)$ (After update)
AB, $c\_update = 2$	AB	AB, $c\_update = 3$
AC, $c\_update = 1$	AC	AC, $c\_update = 2$
AD, $c\_update = 1$		AD, $c\_update = 1$ (X)
BC, $c\_update = 2$	BC	BC, $c\_update = 3$
	BD	BD, $c\_update = 1$ (X)

Figure 3: Update candidate large 2-itemsets for fuzzy match (Example 1)

does not satisfy $c\_update + (N - n) \geq m \cdot N$. Thus, in the algorithm outlined in Figure 2, steps 4 and 13 for fuzzy match can be instantiated by the following procedure.

```
**Procedure Update4FuzzyMatch** ($L_k(e)$, $L_k(e_0)$)
Let $n$ be the number of times that $L_k(e)$ has been updated (including this update);
**if** $L_k(e)$ has never been updated **then**
    Let $L_k(e) = L_k(e_0)$, and set $l.c\_update = 1$ for each $l \in L_k(e)$;
**else**
    set $l.c\_update = 1$ for each $l \in L_k(e_0) - L_k(e)$, and $l.c\_update{+}{+}$ for each $l \in L_k(e_0) \cap L_k(e)$;
    $L_k(e) = \{l \in L_k(e) \cup L_k(e_0) | l.c\_update + (N - n) \geq m \cdot N\}$;
**endif**
```

Example 1 Suppose we are given a calendar schema $R = (\textit{week} : \{1, \cdots, 5\}, \textit{day} : \{1, \cdots, 7\})$ and a fuzzy match ratio $m = 0.8$. Consider the calendar pattern $\langle *, 2\rangle$. There are totally 5 basic time intervals covered by $\langle *, 2\rangle$ (i.e., $N = 5$). Suppose we have computed the large 2-itemsets for the basic time interval $\langle 3, 2\rangle$ and want to update the candidate large 2-itemsets for $\langle *, 2\rangle$. In Figure 3, the set of candidate large 2-itemsets (i.e., $L_2(\langle *, 2\rangle)$) is given in the left column, and the set of large 2-itemsets for $\langle 3, 2\rangle$ (i.e., $L_2(\langle 3, 2\rangle)$) is in the middle column. Suppose this is the third time that $L_2(\langle *, 2\rangle)$ is updated (i.e., $n = 3$). Then the resulting $L_2(\langle *, 2\rangle)$ can be computed as in the last column. The itemsets marked with 'X' do not satisfy the condition $c\_update + (N - n) \geq m \cdot N$ and thus are dropped from $L_2(\langle *, 2\rangle)$. □

If the set of large k-itemsets $L_k(e_0)$ is correctly computed for each basic time interval e_0, then *Update4FuzzyMatch* can correctly generate large k-itemsets w.r.t. fuzzy match for all star calendar patterns. This is guaranteed by the following lemma.

Lemma 1 *Consider the algorithm outlined in Figure 2. If procedure* Update4FuzzyMatch *is used at steps 4 and 13 and $L_k(e_0)$ is the set of large k-itemsets for each basic time interval e_0, then after all the basic time intervals are processed, for each calendar pattern e, $L_k(e)$ contains all and only the k-itemsets that are large for at least* $100m\%$

of the basic time intervals covered by e.

3.2.1 Calendar Tree

In the algorithm, it is necessary to locate the large itemsets for a given calendar pattern quickly. We use a data structure called a *calendar tree* to organize the large itemsets for all the calendar patterns.

Given a calendar schema $R = (f_n : D_n, f_{n-1} : D_{n-1}, \ldots, f_1 : D_1)$, the calendar tree for R is a tree of height n. Itemsets are stored in the leaf nodes. An interior node at height i contains a look-up table of size $|D_i| + 1$, which has one cell for each domain value in D_i plus a cell for the wild-card symbol '*'. Each cell of the look-up table contains a pointer to a node at height $i - 1$. The root is at height n (corresponding to f_n). When we want to locate the leaf node that stores the set of large itemsets for a calendar pattern $e = \langle d_n, d_{n-1}, \ldots, d_1 \rangle$, we start from the root, follow the pointer corresponding to d_n, and from this node follow the pointer corresponding to d_{n-1}, and so on, until we reach the leaf node corresponding to e.

3.3 Generating Candidate Large Itemsets for Precise Match

3.3.1 Direct-Apriori for Precise Match

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 match*, or just *Direct-Apriori* when it is clear from the context. Phase I of *Direct-Apriori* is instantiated as follows.

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

Direct-Apriori for precise match can correctly generate the large k-itemsets w.r.t. precise match. As we discussed earlier (in subsection 3.2), pass 1 of the algorithm can correctly generate the large 1-itemsets for all calendar patterns. Consider a basic time interval e_0 in pass k for $k > 1$. According to *Apriori* [AS94], 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 3.2, for each calendar star pattern e, $L_k(e)$ will consist of the k-itemsets that are large for each basic time interval covered by e after all the basic time intervals are processed.

3.3.2 Temporal-Apriori for Precise Match

Direct-Apriori cannot achieve the best performance; it not only ignores the assumption that we are not interested in temporal association rules for individual basic time intervals, but also the relationship among calendar patterns. Here we present two optimization techniques, which we call *temporal aprioriGen* and *horizontal pruning* respectively, to improve the candidate generation by considering these issues. The resulting algorithm is called *Temporal-Apriori for precise match*, or Temporal-Apriori when it is clear from the context.

The first optimization technique *temporal aprioriGen* is partially based on the assumption mentioned above. Since we are not interested in the large itemsets for basic time intervals, during the processing of each basic time interval e_0, we do not need to count the supports for all the potentially large k-itemsets generated by $C_k(e_0) = aprioriGen(L_{k-1}(e_0))$. 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.

Temporal aprioriGen is also based on an observation about the relationships between the calendar patterns on the same calendar schema. This observation is given in the following lemma.

Lemma 2 *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.*

Lemma 2 gives us an opportunity to improve the generation of candidate large itemsets. Consider the set of candidate large k-itemset for e_0, i.e., $C_k(e_0)$. We only need $C_k(e_0)$ to generate large itemsets for patterns e that cover e_0 (since our data mining problem excludes 0-star patterns in the output). Now we need to have a k-itemset in $C_k(e_0)$ only if it is large for all the 1-star patterns that cover e_0. Indeed, 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 by the above lemma. Thus, using *temporal aprioriGen*, we can generate the candidate large k-itemsets ($k > 1$) via the following procedure.

```
Procedure TemporalAprioriGen4PreciseMatch($e_0$)
$C_k(e_0) = \emptyset$;
forall 1-star patterns $e_1$ that covers $e_0$ do
    $C_k(e_0) = C_k(e_0) \cup$ aprioriGen($L_{k-1}(e_1)$);
return $C_k(e_0)$
```

Example 2 Consider the calendar schema $R = (\mathit{week} : \{1, \cdots, 5\}, \mathit{day} : \{1, \cdots, 7\})$. Suppose we have computed 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 $L_2(\langle 3, *\rangle) = \{AB, AC, AD, BD, CD\}$. If we use *temporal aprioriGen* to compute the candidate large 3-itemsets, 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 2, but applied during a pass. 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 2, l cannot be large for e. Therefore, we may drop l from $C_k(e_0)$. In summary, *Horizontal pruning* can be implemented by the following procedure.

```
**Procedure HorizontalPrune4PreciseMatch**($C_k(e_0), e_0$)
**if** there exists a 1-star pattern $e_1$ that covers $e_0$ such that $L_k(e_1)$ has not been updated even once
    **then return** $C_k(e_0)$;
$P = \emptyset$;
**forall** 1-star patterns $e_1$ that covers $e_0$ **do**
    $P = P \cup L_k(e_1)$;
**return** $(C_k(e_0) \cap P)$.
```

Example 3 Let us continue example 2. Suppose when the basic time interval $\langle 3, 2\rangle$ is being processed, we already have $L_3(\langle *, 2\rangle) = \{ABD\}$ and $L_3(\langle 3, *\rangle) = \{ABD, ACD\}$. Given the generated set of candidate large 3-itemsets $C_3(\langle 3, 2\rangle) = \{ABC, ABD, ACD\}$, we can further prune it by $C_3(\langle 3, 2\rangle) = C_3(\langle 3, 2\rangle) \cap (L_3(\langle *, 2\rangle) \cup L_3(\langle 3, *\rangle)) = \{ABD, ACD\}$. □

In summary, phase I of Temporal-Apriori for precise match can be instantiated as follows.

```
$C_k(e_0)$ = TemporalAprioriGen4PreciseMatch($e_0$);
$C_k(e_0)$ = HorizontalPrune4PreciseMatch($C_k(e_0)$, $e_0$);
```

We prove the correctness of Temporal-Apriori for precise match in the following way. First, we show that the algorithm has the same output as Direct-Apriori if for each basic time interval 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. This result is summarized in Lemma 3 and Theorem 1.

Lemma 3 *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 basic time interval 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.*

	day 1	day 2	day 3	day 4	day 5	day 6	day 7
week 1	X	X	X	X			X
week 2		X	X	X	X	X	
week 3	X	X	X	X	X	X	X
week 4		X	X	X	X	X	X
week 5	X	X	X	X	X	X	

Figure 4: Distribution of large itemset l

3.4 Generating Candidate Large Itemsets for Fuzzy Match

3.4.1 Direct-Apriori for Fuzzy Match

As we discussed earlier, given a fuzzy match ratio m, an itemset is large for a calendar pattern e w.r.t. fuzzy match if it is large for at least $100m\%$ of the basic time intervals covered by e. For brevity, we still refer to such itemsets as large itemsets in the context of fuzzy match.

Similar to Direct-Apriori for precise match, *Apriori*'s candidate generation method can be directly applied to each individual basic time interval when fuzzy match is considered. We call this approach *Direct-Apriori for fuzzy match*. Then phase I of Direct-Apriori for fuzzy match is instantiated in the same way as for precise match, i.e., $C_k(e_0) = aprioriGen(L_{k-1}(e_0))$.

Indeed, Direct-Apriori supplies phase III (i.e., procedure Update4FuzzyMatch) with the set of large k-itemsets for each basic time interval. By Lemma 1, it is easy to see that Direct-Apriori for fuzzy match can correctly generate large itemsets w.r.t. fuzzy match for all calendar patterns.

3.4.2 Temporal-Apriori for Fuzzy Match

Recall that our first optimization technique *temporal aprioriGen* is based on both Lemma 2 and the assumption that we are not interested in large itemsets for basic time intervals. The assumption still applies when fuzzy match is considered. However, Lemma 2 is not true in the context of fuzzy match. This can be seen through a counter example.

Example 4 Consider a calendar schema $R = (\textit{week} : \{1, \cdots, 5\}, \textit{day} : \{1, \cdots, 7\})$ and fuzzy match ratio $m = 0.8$. Suppose an itemset l is large for the basic time intervals marked with 'X' in Figure 4. The figure shows that l is large for $\langle 1, 1\rangle$. Moreover, l is large for the calendar pattern $\langle *, *\rangle$ since it is large for 29 basic time intervals (the match ratio $m = 0.8$ requires l is large for at least $0.8 \cdot 5 \cdot 7 = 28$ basic time intervals). However, l is not large in neither $\langle *, 1\rangle$ nor $\langle 1, *\rangle$. For $\langle *, 1\rangle$, l is large for 3 basic time intervals, which is less than $0.8 \cdot 5 = 4$. For $\langle 1, *\rangle$, l is large for 5 basic time intervals, which is less than $0.8 \cdot 7 = 5.6$. □

Example 4 shows that an itemset may be large for a star calendar pattern e even if it is not large for any 1-star pattern covered by e. Therefore, we cannot directly use the same approach as we did for precise match. Nevertheless, we can still apply *temporal aprioriGen* to fuzzy match with some modification: We just consider all star calendar patterns instead of 1-star calendar patterns. This is correct since if a k-itemset is large for a calendar pattern e, then each of its $(k-1)$-item subset must be large for e. The procedure is shown as follows.

```
**Procedure TemporalAprioriGen4FuzzyMatch**($e_0$)
$C_k(e_0) = \emptyset$;
**forall** star patterns $e$ that covers $e_0$ **do**
    $C_k(e_0) = C_k(e_0) \cup aprioriGen(L_{k-1}(e) \cap L_{k-1}(e_0))$ (*);
return $C_k(e_0)$.
```

Example 5 Consider 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 $L_2(\langle *, *\rangle) = \{AB, AD, BD, CD, AC, AE\}$. If we use Temporal-Apriori for fuzzy match to compute the candidate large 3-itemsets, 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 $C_3(\langle 3, 2\rangle) = C_3(\langle *, 2\rangle) \cup C_3(\langle 3, *\rangle) \cup C_3(\langle *, *\rangle) = \{ABD, ACD, ACE\}$. □

Note that in the procedure *TemporalAprioriGen4FuzzyMatch*, the statement marked with "*" requires access to the large $(k-1)$-itemsets for basic time intervals. When the calendar schema includes a large number of basic time intervals, this step will greatly increase the memory requirement, since the large itemsets for these basic time intervals must be kept. An alternative is to use $L_{k-1}(e)$ instead of $L_{k-1}(e) \cap L_{k-1}(e_0)$ when memory is the critical resource. This alternative can reduce the memory requirement by not keeping all the large itemsets for basic time intervals; however, the downside is that it may generate some candidates that would not even be generated by Direct-Apriori. In our experiments, we use the original proposal for pass 2 and the alternative way for the later passes.

Due to the difference between precise match and fuzzy match, our second optimization technique for precise match, *horizontal pruning*, cannot be directly applied to fuzzy match, either. This is because fuzzy match allows a large itemset to be not large 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.

This observation leads to the following pruning procedure. Note that we reuse the procedure *Update4FuzzyMatch*, which was developed to update the large itemsets w.r.t. fuzzy match (see subsection 3.2). The idea is to discard the candidate large itemsets that cannot be large for calendar pattern e even if they are large for e_0.

```
Procedure HorizontalPrune4FuzzyMatch($C_k(e_0), e_0$)
if there exists $e$ that covers $e_0$ such that $L_k(e)$ has not been updated
    then return $C_k(e_0)$;
$P = \emptyset$;
forall star patterns $e$ that covers $e_0$ do begin
    $C_k(e) = L_k(e)$;
    Update4FuzzyMatch($C_k(e), C_k(e_0)$);
    $P = P \cup C_k(e)$;
end
return $(C_k(e_0) \cap P)$.
```

Example 6 Let us continue example 5. We have generated a set of candidate large 3-itemsets $C_3(\langle 3,2\rangle) = \{ABD$, ACD, $ACE\}$. Suppose all of $L_3(\langle *,2\rangle)$, $L_3(\langle 3,*\rangle)$, and $L_3(\langle *,*\rangle)$ have been updated at least once. Assuming that all itemsets in $C_3(\langle 3,2\rangle)$ were large for $\langle 3,2\rangle$, we can use the procedure *Update4FuzzyMatch* to 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 $C_3(\langle 3,2\rangle) = C_3(\langle 3,2\rangle) \cap (C_3(\langle *,2\rangle) \cup C_3(\langle 3,*\rangle) \cup C_3(\langle *,*\rangle)) = \{ABD, ACD\}$. □

Using the fuzzy match version of *temporal aprioriGen* and *horizontal pruning*, phase I of Temporal-Apriori can be instantiated as follows.

$C_k(e_0)$ = TemporalAprioriGen4FuzzyMatch(e_0);
$C_k(e_0)$ = HorizontalPrune4FuzzyMatch($C_k(e_0)$, e_0);

The correctness of Temporal-Apriori for fuzzy match can be shown in the same way as for precise match. First, we show that the algorithm has the same output as Direct-Apriori for fuzzy match if for each basic time interval e_0, it uses a super set of the union of large k-itemsets for all calendar patterns covering 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. This result is summarized in Lemma 4 and Theorem 2.

Lemma 4 *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 basic time interval 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 on a DELL OptiPlex GX200 PC running Windows 2000 Professional. The PC has a 667 MHz Pentium III CPU with 256

pass k	Precise match		Fuzzy match ($m = 0.9$)		Fuzzy match ($m = 0.8$)	
	# calendar patterns	# large k-itemsets	# calendar patterns	# large k-itemsets	# calendar patterns	# large k-itemsets
2	130	3812	130	4003	130	4472
3	130	1770	130	1868	130	2179
4	92	341	103	366	122	445
5	28	29	29	30	32	33

Figure 5: Discovery in the KDD Cup 2000 data ($minsupport = 0.75\%$)

KB full cache and 256 MB main memory. The data sets were stored on a 30 GB, 7200 RPM EIDE hard disk.

In the following, we first assess the performance of our algorithms using the transactional data published in KDD Cup 2000 [KB00]. Then we generate synthetic data sets to further evaluate the algorithms with data sets having various characteristics.

4.1 KDD Cup 2000 Data Set

We choose the clicks data file in the KDD Cup 2000 data sets to perform our experiments. The clicks data file consists of homepage request records, each of which contains attribute values describing the request and the person who sent the request. Examples of the attributes include when the request was submitted, where the person lives, and how many children the person has, and so on. We consider each request record as a transaction.

The requests recorded in the clicks data file are from January 30, 2000 to March 31, 2000, which cover 8 weeks (from the 6th to the 13th week in year 2000) plus 6 days (in the 14th week). We use *timeOfDay* to represent the calendar concept formed by partitioning each day into three parts: *early morning* (0am - 8am), *daytime* (8am - 4pm), and *evening* (4pm - 12pm). We use the calendar schema $R_{KDD2K} = (\texttt{week} : \{6, 7, \cdots, 14\}, \texttt{day} : \{1, 2, \cdots, 7\}, \texttt{timeOfDay} : \{1, 2, 3\})$, where the domain values of *week* represent the number of week of year 2000, the domain values of *day* represent *Sunday, Monday,* $\cdots$, *Saturday*, the domain values of *timeOfDay* represent *early morning*, *daytime*, and *evening*. The predicate *valid* evaluates to True for all basic time intervals between January 30, 2000 and March 31, 2000.

We preprocess the clicks data file to remove NULL and unknown values marked with ’?’. To simplify the problem, we focus on the categorical attributes and ignore all the attributes identified as “ignore”, “date”, “time”, and “continuous”. The preprocessed data set consists of 777,480 transactions. The largest transaction consists of 100 items, the smallest transaction consists of 5 items, and the transactions contain 23.4 items on average. Using the aforementioned calendar schema R_{KDD2K}, the maximum and the minimum number of transactions per basic time interval are 27,807 and 12, respectively, and the average number of transactions per basic time interval is 4,180.

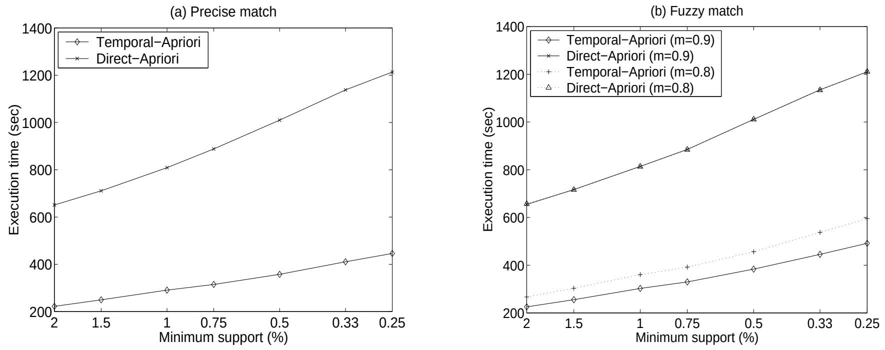

Figure 6: Execution time of our algorithms on the KDD Cup 2000 data

The experimental results are summarized in Figures 5 and 6. Figure 5 shows the number of calendar patterns and large itemsets discovered from the data set with the minimum support $minsupport = 0.75\%$. We discovered up to large 5-itemsets along with their calendar patterns. It is also interesting to note that we indeed discovered more patterns with fuzzy match than with precise match, and the smaller the match ratio we used for fuzzy match, the more patterns we discovered.

Figure 6 shows the execution time of Direct-Apriori and Temporal-Apriori w.r.t. both precise match and fuzzy match with match ratios 0.9 and 0.8. (Note that in Figure 6(b), Direct-Apriori for fuzzy match took almost the same time for match ratios 0.9 and 0.8.) The result shows that our optimization techniques improve the performance by 2 to 3 times. However, since the size of the data set is not very large (777,480 transactions with 23.4 items per transaction on average), the operating system can cache all the data in physical memory. In the next subsection, we validate this performance gain for very large data sets.

4.2 Synthetic Data Sets

In order to generate data sets with various characteristics, we extend the transaction data generator proposed in [AS94] to incorporate temporal features. Specifically, 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 [AS94]. (Due to space reason, we do not repeat this method in the paper; the reader is referred to [AS94] for the details.)

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* that are shared across 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.

Notation	Meaning	Default value
$\lvert D \rvert$	Number of transactions per basic time interval	10,000
$\lvert T \rvert$	Avg. size of the transactions	10
$\lvert I \rvert$	Avg. size of the maximal potentially large itemsets	4
$\lvert L \rvert$	Avg. number of the maximal potentially large itemsets per basic time interval	1,000
N	Number of items	1,000
C	Calendar Schema	(year:{1995-1999},month,day)
P_r	Pattern-ratio	0.4
N_p	Avg. number of star calendar patterns per pattern itemset	40

Figure 7: Parameters for data generation

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 (i.e., $P_r \times$ the total number of per-interval itemsets). Each time when a pattern itemset is chosen, we use it as a per-interval itemset if it has an associated calendar pattern that covers the basic time interval; otherwise, the itemset is ignored. Intuitively, the more calendar patterns are assigned to a pattern itemset, the more chances that the pattern itemset is used as per-interval itemsets. 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. The calendar pattern to be assigned to a pattern itemset is then chosen by tossing an $|\mathcal{P}|$-sided weighted coin, where $|\mathcal{P}|$ is the total number of calendar patterns.

Our data generation procedure takes eight parameters, which are shown in Figure 7. The upper part of table shows the parameters required by the original data generator proposed in [AS94], while the lower part shows the parameters related to temporal features. Figure 7 also shows the default values of the parameters. To examine the performance of the algorithms with data sets having different characteristics, we generated a series of data sets, most of which were generated 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.

Our first set of experiments was to evaluate the optimization techniques with synthetic data sets. We generated

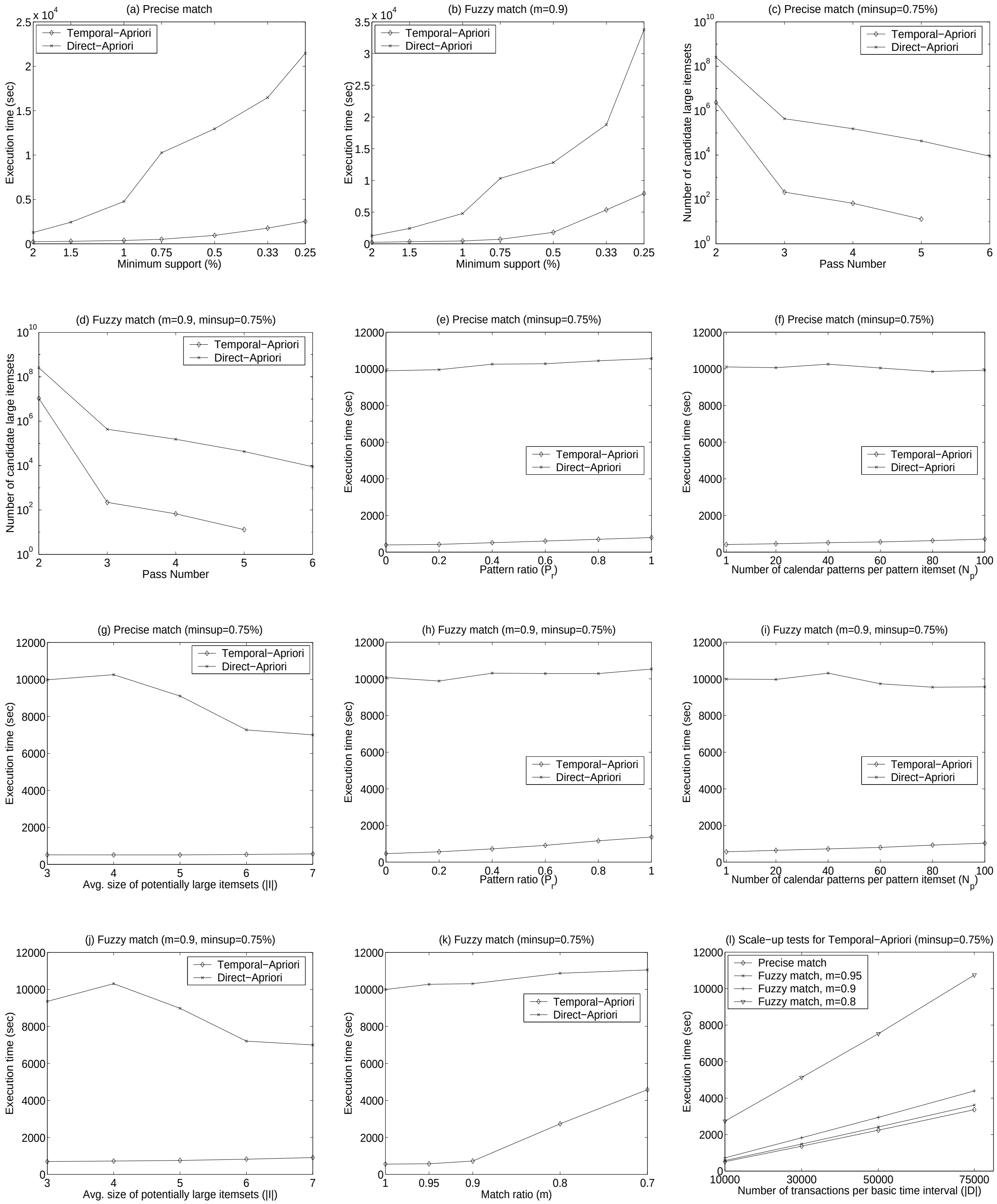

Figure 8: Experimental result on synthetic data sets

a data set using the default parameters and performed experiments with various minimum supports for both precise match and fuzzy match (match ratio $m = 0.9$). The execution time is shown in Figures 8(a) and 8(b). The experimental result shows that our optimization techniques are quite effective. 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. Moreover, all the algorithms are sensitive to the minimum support: the smaller the minimum support is, the longer the execution time is. However, the execution time of Temporal-Apriori increases much slower than that of Direct-Apriori. Figures 8(c) and 8(d) also give the total number of candidate large itemsets for the experiments with the minimum support 0.75%, showing that our two optimization techniques greatly reduced the number of candidates in each pass.

Our second set of experiments was intended to evaluate the performance of both Temporal-Apriori and Direct-Apriori with various kinds of data sets. We generated three sets of data sets. The first set of data sets uses different values for P_r and default values for other parameters. Similarly, the second and the third set of data sets uses different values for N_p and $|I|$, respectively, and uses default values for other parameters. The experiments were performed using the minimum support 0.75%. Figures 8(e) through 8(j) show the execution time of both precise match and fuzzy match for these data sets. In all the experiments, Temporal-Apriori performs significantly better than Direct-Apriori. Figures 8(e) and 8(h) indicate that Direct-Apriori is not very sensitive to pattern ratio. However, the execution time of Temporal-Apriori increases by 100% for precise match and 200% for fuzzy match as the pattern ratio P_r ranges from 0 to 1. The reason is that when the pattern ratio P_r increases, the number of large itemsets that have temporal patterns increases as well and thus results in a larger number of candidate large itemsets. In Figures 8(f) and 8(i), the execution time of Temporal-Apriori increases by 75% for both precise and fuzzy match when the parameter N_p (i.e., the number of calendar patterns per pattern itemset) ranges from 1 to 100. This is because a larger N_p increases the supports of itemsets and results in a larger set of candidates. In contrast, the execution time of Direct-Apriori decreases slightly as N_p increases. Figures 8(g) and 8(j) show that the execution time of Temporal-Apriori increases slightly as the average size of potentially large itemsets ($|I|$) ranges from 3 to 7, while the execution time of Direct-Apriori decreases by about 20%.

Our third set of experiments was intended to study the impact of match ratio to our fuzzy-match algorithms. Figure 8(k) shows the execution time of both Temporal-Apriori and Direct-Apriori with various match ratios. These experiments used the data set with all default parameters and the minimum support 0.75%. The result indicates that Temporal-Apriori is very sensitive to match ratio: the larger the match ratio is, the longer time Temporal-Apriori takes. The execution time of Direct-Apriori also increases slightly. Nevertheless, in the worst case when match ratio is 0.7, Temporal-Apriori still performs significantly better than Direct-Apriori.

Finally, to examine the scalability of Temporal-Apriori, we generated a series of data sets with increasing number of transactions per basic time interval and performed a set of experiments for precise match and fuzzy match with different match ratios. The sizes of the data sets range from 739MB to 5.41 GB. As shown in Figure 8, Temporal-Apriori takes time linear to the number of transactions.

In summary, our experiments on synthetic data sets show that our optimization techniques are quite effective and the algorithms are stable for various kinds of data sets. In addition, our optimized algorithm scales up very well w.r.t. the number of transactions.

5 Related Work

Since the concept of association rule was first introduced in [AIS93], discovery of association rules has been extensively studied [AS94, SON95, BMUT97, ZPOL97, AS96, HKK97, SK98]. The concept of association rule was also extended in several ways, including generalized rules and multi-level rules [SA95, HF95], multi-dimensional rules, quantitative rules [SA96, MY97], and constraint-based rules [BAG99, NLHM99]. Among these extensions is the discovery of temporal association rules.

There are several kinds of meaningful temporal association rules. The problem of mining cyclic association rules (i.e., the association rules that occur periodically over time) has been studied in [ÖRS98]. Several algorithms and optimization techniques were presented in [ÖRS98] and shown effective through a series of experiments. 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 [RMS98], the work in [ÖRS98] was further extended to approximately discover user-defined temporal patterns in association rules. The work in [RMS98] is more flexible and practical than [ÖRS98]; 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. Although the calendar algebra adopted in [RMS98] is a powerful tool to define temporal patterns, users need to know exactly what temporal patterns they are interested in to give such expressions. In some cases, users lack such prior knowledge.

Our work differs from [ÖRS98] and [RMS98] 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 [ÖRS98] and [RMS98]. In addition, unlike [RMS98], 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. Finally, based on the representation mechanisms proposed in [LMF86] or [BJW00], we can have calendar schemas for both cyclic and user-defined temporal patterns. Thus, cyclic patterns and calendar algebra expressions can be considered as special cases of calendar patterns.

In [AR00], the discovery of association rules that hold in the transactions during the items' life time was discussed. The algorithm *Apriori* was extended to discover such association rules. Our problem differs in that we consider the association rules for calendar patterns instead of the life time of the items.

There are other related research activities. In [LWJ00], discovery of calendar-based event patterns was discussed.

In [CP98], a generic definition of temporal patterns and a framework for discovering them were presented. In [CP99], the discovery of the longest intervals and the longest periodicity of association rules was discussed. In [RR99], it was proposed to add temporal features to association rules by associating a conjunction of binary temporal predicates that specify the relationships between the timestamps of transactions. These works consider different aspects of temporal data mining; we consider them as complementary to ours. Finally, a bibliography of temporal data mining can be found in [RS99].

6 Conclusion and Future Work

In this paper, we studied the discovery of association rules along with their temporal patterns in terms of calendar schemas. We identified 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. An important feature of our representation mechanism is that the corresponding data mining problem requires less prior knowledge than the prior methods and hence may discover more unexpected rules. The discovered rules are easier to understand. Moreover, we extended *Apriori*, an existing algorithm for mining association rules, to discover temporal association rules w.r.t. both precise match and fuzzy match. By studying the relationships among calendar patterns, we developed two optimization techniques to improve the performance of the data mining process. Our experiments showed that our optimization techniques are quite effective.

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

[AIS93] 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.

[AR00] J.M. Ale and G.H. Rossi. An approach to discovering temporal association rules. In *Proc. of the 2000 ACM Symposium on Applied Computing*, pages 294–300, 2000.

[AS94] 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.

[AS96] R. Agrawal and J. C. Shafer. Parallel mining of association rules. *IEEE Transactions on Knowledge and Data Engineering*, 8(6):962–969, 1996.

[BAG99] R.J. 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.

[BJW00] C. Bettini, S. Jajodia, and X.S. Wang. *Time granularities in databases, data mining, and temporal reasoning*. Springer-Verlag, 2000.

[BMUT97] S. Brin, R. Motwani, J. D. Ullman, and S. Tsur. Dynamic itemset counting and implication rules for market basket data. In *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, pages 255–264, 1997.

[CP98] X. Chen and I. Petrounias. A framework for temporal data mining. In *Proc. of the 9th Int'l Conf. on Database and Expert Systems Applications*, pages 796–805, 1998.

[CP99] 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.

[HF95] 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.

[HKK97] E. Han, G. Karypis, and V. Kumar. Scalable parallel data mining for association rules. In *Proc. of the 1997 ACM SIGMOD Int'l Conf. on Management of Data*, pages 277–288, 1997.

[KB00] 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/.

[LMF86] B. Leban, D. McDonald, and D. Foster. A representation for collections of temporal intervals. In *Proc. of AAAI-1986 5th Int'l Conf. on Artifical Intelligence*, pages 367–371, 1986.

[LWJ00] Y. Li, X.S. Wang, and S. Jajodia. Discovering temporal patterns in multiple granularities. In *Proc. of Int'l Workshop on Temporal, Spatial and Spatio-temporal Data Mining*, 2000.

[MY97] R. J. Miller and Y. Yang. Association rules over interval data. In *Proc. of the 1997 ACM SIGMOD Int'l Conf. on Management of Data*, pages 452–461, 1997.

[NLHM99] R. T. Ng, L. V. S. Lakshmanan, J. Han, and T. Mah. Exploratory mining via constrained frequent set queries. In *Proc. of the 1999 ACM SIGMOD Int'l Conf. on Management of Data*, pages 556–558, 1999.

[ÖRS98] 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.

[RMS98] 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.

[RR99] C.P. Rainsford and J.F. 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.

[RS99] J.F. Roddick and M. Spiliopoulou. A bibliography of temporal, spatial and spatio-temporal data mining research. *SIGKDD Explorations*, 1(1):34–38, June 1999.

[SA95] R. Srikant and R. Agrawal. Mining generalized association rules. In *Proc. of the 21th Int'l Conf. on Very Large Data Bases*, pages 407–419. Morgan Kaufmann, 1995.

[SA96] 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.

[SK98] T. Shintani and M. Kitsuregawa. Parallel mining algorithms for generalized association rules with classification hierarchy. In *Proc. of ACM SIGMOD Int'l Conf. on Management of Data*, pages 25–36, 1998.

[SON95] A. Savasere, E. Omiecinski, and S. B. Navathe. An efficient algorithm for mining association rules in large databases. In *Proc. of the 1995 Int'l Conf. on Very Large Data Bases*, pages 432–444, 1995.

[ZPOL97] M.J. Zaki, S. Parthasarathy, M. Ogihara, and W. Li. New algorithms for fast discovery of association rules. In *Proc. of the 3rd Int'l Conf. on Knowledge Discovery and Data Mining*, pages 283–286, 1997.

A Proof Sketch

Proof of Lemma 1 Consider any calendar pattern e. Suppose N basic time intervals are covered by e. For each itemset in $L_k(e)$, its counter $c\_update \geq m \cdot N$ since it is not dropped in the last update. For each itemset dropped in the n-th update of $L_k(e)$, its counter $c\_update \leq c\_update + (N - n) < m \cdot N$, i.e., it cannot be large for more than $100m\%$ of the basic time intervals covered by e. Since all large itemsets are processed, for all calendar patterns e, $L_k(e)$ contains all and only the itemsets that are large for at least $100m\%$ of the basic time intervals covered by e. □

Proof of Lemma 2 Suppose there exists a 1-star calendar pattern e_1 covered by e such that an itemset l is not large for e_1 w.r.t. precise match. Then there exists at least one basic time interval e_0 covered by e_1 for which l is not large. Since e_1 is covered by e, the basic time interval e_0 is also covered by e. Then l is not large for at least one basic time interval e_0 covered by e, which leads to contradiction. □

Proof of Lemma 3 It suffices to prove that for each pass k, if Temporal-Apriori uses a super set of $\bigcup_{e_1 \ covers \ e_0} L_k(e_1)$ as the set of candidate large k-itemsets for e_0, it has the same output as Direct-Apriori for precise match.

Consider the algorithm Direct-Apriori. Denote the set of candidate large k-itemsets generated in phase I as $C_k(e_0)$, the set of large k-itemsets generated in phase II as $L_k(e_0)$, and the output for each star calendar pattern e as $L_k(e) =$

$\bigcap_{e_0 \ covered \ by \ e} L_k(e_0)$ in output.

Consider the algorithm Temporal-Apriori. Denote the set of candidate large k-itemsets, which is a super set of $\bigcup_{e_1 \ covers \ e_0} L_k(e_1)$, as $C'_k(e_0)$, the large itemsets derived from $C'_k(e_0)$ in phase II as $L'_k(e_0)$, and the output for each star calendar pattern e as $L'_k(e) = \bigcap_{e_0 \ covered \ by \ e} L'_k(e_0)$. We need to prove $L'_k(e) = L_k(e)$ for each star pattern e.

Since $L_k(e_0)$ is the set of all large k-itemsets in $\mathcal{T}[e_0]$ by definition, it is easy to see $L'_k(e_0) \subseteq L_k(e_0)$ for all e_0. Thus, we have $L'_k(e) \subseteq L_k(e)$.

Now let's prove $L_k(e) \subseteq L'_k(e)$. Given a basic time interval e_0, let $L''_k(e_0) = L_k(e_0) \cap (\bigcup_{e_1 \ covers \ e_0} L_k(e_1))$ and $L''_k(e) = \bigcap_{e_0 \ covered \ by \ e} L''_k(e_0)$. Since $C'_k(e_0)$ is a super set of $\bigcup_{e_1 \ covers \ e_0} L_k(e_1)$, $C'_k(e_0)$ is also a super set of $L''_k(e_0)$. When $L'_k(e_0)$ is computed from $C'_k(e_0)$, all k-itemsets in $L''_k(e_0)$ remain in $L'_k(e_0)$ since $L''_k(e_0) = L_k(e_0) \cap (\bigcup_{e_1 \ covers \ e_0} L_k(e_1))$. Thus, we have $L''_k(e_0) \subseteq L'_k(e_0)$ and then $L''_k(e) \subseteq L'_k(e)$.

By definition, for each $l \in L_k(e)$, l is in $L_k(e_0)$ for all e_0 covered by e. By lemma 2, l is also in $L_k(e_1)$ for all e_1 covered by e. It is easy to see that if e_0 is covered by e, then at least one 1-star calendar pattern e_1 that covers e_0 is also covered by e. It follows that for all e_0 covered by e, l is in $L''_k(e_0)$, i.e., $l \in L''_k(e)$. This shows $L_k(e) \subseteq L''_k(e)$. Consider the fact $L''_k(e) \subseteq L'_k(e)$, we have $L_k(e) \subseteq L'_k(e)$. This concludes the proof. □

Proof of Theorem 1 First, the set of candidate large k-itemsets generated by *TemporalAprioriGen* is a super set of $\bigcup_{e_1 \ covers \ e_0} L_k(e_1)$, since for each 1-star calendar pattern e_1 that covers e_0, *aprioriGen* generates a super set of $L_k(e_1)$. Second, if the input of *HorizontalPrune* is a super set of $\bigcup_{e_1 \ covers \ e_0} L_k(e_1)$, then its output is also a super set, since the output is the intersection of the input and $\bigcup_{e_1 \ covers \ e_0} L_k(e_1)$. By Lemma 3, we know Temporal-Apriori has the same output as Direct-Apriori. That is, Temporal-Apriori is equivalent to Direct-Apriori. □

Proof of Lemma 4 and Theorem 2 Lemma 4 and Theorem 2 can be proved in the same way as Lemma 3 and Theorem 2, respectively. Proofs are omitted due to space reason. □