Mining General Temporal Association Rules for Items with Different Exhibition Periods

Cheng-Yue Chang, Ming-Syan Chen and Chang-Hung Lee
Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, ROC
Email: {cychang, chlee}@arbor.ee.ntu.edu.tw; mschen@cc.ee.ntu.edu.tw

Abstract

In this paper, we explore a new model of mining general temporal association rules from large databases where the exhibition periods of the items are allowed to be different from one to another. Note that in this new model, the downward closure property which all prior Apriori-based algorithms relied upon to attain good efficiency is no longer valid. As a result, how to efficiently generate candidate itemsets form large databases has become the major challenge. To address this issue, we develop an efficient algorithm, referred to as algorithm SPF (standing for Segmented Progressive Filter) in this paper. The basic idea behind SPF is to first segment the database into sub-databases in such a way that items in each sub-database will have either the common starting time or the common ending time. Then, for each sub-database, SPF progressively filters candidate 2-itemsets with cumulative filtering thresholds either forward or backward in time. This feature allows SPF of adopting the scan reduction technique by generating all candidate k-itemsets ($k > 2$) from candidate 2-itemsets directly. The experimental results show that algorithm SPF significantly outperforms other schemes which are extended from prior methods in terms of the execution time and scalability.

1. Introduction

In recent years, a significant amount of research effort has been elaborated upon deriving data mining techniques to discover useful but unknown knowledge from large databases. The knowledge discovered includes those on association rules, classification rules, sequential patterns, path traversal patterns, user moving patterns, and etc. Among others, mining of association rules is a well addressed important problem, i.e., "Given a database of sales transactions, one would like to discover the important associations among items such that the presence of some items in a transaction will imply the presence of other items in the same transaction."

The problem of mining association rules was first explored by [2]. Following this pioneering work, several fast algorithms based on the level-wise Apriori framework [3, 15] and partitioning [12] are proposed to remedy the performance bottleneck of Apriori. In addition, several novel mining techniques, including TreeProjection [1] and FP-growth algorithms [10], also receive a significant amount of research attention. On the other hand, many variants of mining association rules are studied to explore more mining capabilities, such as incremental updating [5, 11], mining of generalized and multi-level rules [8, 16], mining of quantitative rules [17], mining of multi-dimensional rules [14], constraint-based rule mining [9] and mining with multiple minimum supports [13, 18], mining associations among correlated or infrequent items [7], and temporal association rule discovery [4, 6].

While these are important results toward enabling the integration of association mining and fast searching algorithms, their mining methods, however, cannot be effectively applied to the transaction database where the exhibition periods of the items are different from one to another. As a matter of fact, it is a common phenomenon that the items in a real transaction database have different exhibition periods. The problem can be best understood by the following example.

Example 1.1: Consider the transaction database $\mathcal{D}$ as shown in Figure 1. A set of time series database indicates the transaction records from January 2002 to April 2002. The exhibition period of each item is given in the right of Figure 1. Assume that the minimum support and the minimum confidence required are $min\_supp = 30\%$ and $min\_conf = 75\%$, respectively. By conventional mining algorithms, with respect to the same support counting

		Transaction Database		
		Date	TID	Itemset
D	P1	Jan-02	T01	A C F
			T02	A D F
			T03	C F
			T04	A D
	P2	Feb-02	T05	BCD F
			T06	C
			T07	BCD
			T08	AB
	P3	Mar-02	T09	C EF
			T10	AB F
			T11	EF
			T12	B
	P4	Apr-02	T13	A E
			T14	B E
			T15	AB
			T16	AB

Item	start	end
A	Jan-02	Apr-02
B	Feb-02	Apr-02
C	Jan-02	Mar-02
D	Jan-02	Feb-02
E	Mar-02	Apr-02
F	Jan-02	Mar-02

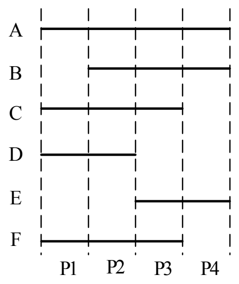


Figure 1. An illustrative database where the items have individual exhibition periods.

basis $|\mathcal{D}|$, only the itemsets $\{A\}$, $\{B\}$, $\{C\}$ and $\{F\}$ will be termed as frequent itemsets. Thus, no rule will be discovered in this case. However, as will be shown later in Example 1.2, some rules do exist in the illustrative database when the individual exhibition periods of items are taken into consideration.

From Example 1.1, we can observe that the problem of conventional mining algorithms lies in their absence of equitable support counting basis for each itemset. It is noted that the itemsets with longer exhibition periods (e.g., A, B, C and F) are more likely to be frequent than those with shorter exhibition periods (e.g., D and E). As a result, the association rules we usually discover will be those composed of long-term items (e.g., milk and bread are frequently purchased together, which is, however, of less interest to us). In contrast, some short-term itemsets, such as those composed of seasonal food or clothes, which may be really "frequent" and interesting in their exhibition periods are less likely to be identified as frequent ones if a conventional mining process is employed. To address this issue, we explore in this paper a new model of mining general temporal association rules where the items are allowed to have different exhibition periods, and the determination of their supports is made in accordance with their exhibition periods. Explicitly, we introduce the notion of *maximal common exhibition period* (abbreviated as *MCP)* and define the *relative support* to provide an equitable support counting basis for each itemset. The *MCP* of the itemset X, denoted by $[p, q]$, is defined as the period between the *latest-exhibition-start time* p and the *earliest-exhibition-end time* q of all items belonging to X. For example, for the itemset BC in Figure 1, its *MCP* is $[2, 3]$ since the *latest-exhibition-start time* of the items B and C is 2 and the *earliest-exhibition-end time* of the items B and C is 3. The relative support of the itemset X is given by $supp(X^{p,q}) = \frac{|\{T \in \mathcal{D}^{p,q} | X \subseteq T\}|}{|\mathcal{D}^{p,q}|}$ where $\mathcal{D}^{p,q}$ indicates the partial database bounded by $[p, q]$, and $|\{T \in \mathcal{D}^{p,q} | X \subseteq T\}|$ indicates the number of transactions that contain X in $\mathcal{D}^{p,q}$. The general temporal association rule $(X \Longrightarrow Y)^{p,q}$ is termed to be frequent within its *MCP* $[p, q]$ if and only if its relative support is not smaller than the minimum support required (i.e., $supp((X \cup Y)^{p,q}) \geq min\_supp$), and its confidence is not smaller than the minimum confidence needed (i.e., $conf((X \Longrightarrow Y)^{p,q}) = \frac{supp((X \cup Y)^{p,q})}{supp(X^{p,q})} \geq min\_conf$).

Example 1.2: Based on the definitions above, the frequent general temporal association rules in Example 1.1 can be identified as follows:

(1) $(A \Longrightarrow B)^{2,4}$ with $supp(AB^{2,4}) = \frac{4}{12} > 30\%$ and $conf((A \Longrightarrow B)^{2,4}) = \frac{supp(AB^{2,4})}{supp(A^{2,4})} = \frac{4}{5} > 75\%$;

(2) $(D \Longrightarrow B)^{2,2}$ with $supp(BD^{2,2}) = \frac{2}{4} > 30\%$ and $conf((D \Longrightarrow B)^{2,2}) = \frac{supp(BD^{2,2})}{supp(D^{2,2})} = \frac{2}{2} > 75\%$;

(3) $(E \Longrightarrow F)^{3,3}$ with $supp(EF^{3,3}) = \frac{2}{4} > 30\%$ and $conf((E \Longrightarrow F)^{3,3}) = \frac{supp(EF^{3,3})}{supp(E^{3,3})} = \frac{2}{2} > 75\%$;

(4) $(D \Longrightarrow BC)^{2,2}$ with $supp(BCD^{2,2}) = \frac{2}{4} > 30\%$ and $conf((D \Longrightarrow BC)^{2,2}) = \frac{supp(BCD^{2,2})}{supp(D^{2,2})} = \frac{2}{2} > 75\%$.

It is important to note that in this new model, the downward closure property which all prior Apriori-based algorithms relied upon to attain good efficiency is no longer valid. Recall that the downward closure property guarantees that all sub-itemsets of a frequent itemset are frequent. Based on this property, all prior Apriori-based algorithms are allowed to limit their attention to those candidates whose sub-itemsets are frequent and to prune the searching space effectively. However, once this property is not valid anymore, the searching space will explode and become difficult to tackle.

Example 1.3: From the illustrative database in Example 1.1, we can find that although the itemset BCD is frequent in its maximal common exhibition period, the itemsets BC, BD and CD are not all frequent in their corresponding maximal common exhibition periods. Explicitly, the itemset BC is infrequent in its maximal common exhibition period $[2, 3]$ since its relative support is only 25% ($< 30\%$). Similarly, the itemset CD is infrequent in its maximal common exhibition period $[1, 2]$. Hence, to determine if an itemset is frequent, without the downward closure property, we are not allowed anymore to limit our attention to those whose subitemsets are frequent.

To address this issue, we develop an efficient algorithm, referred to as algorithm SPF (standing for *Segmented Progressive Filter)* in this paper. In essence, algorithm SPF consists of two major procedures, i.e., *Segmentation* (abbreviated as *ProcSG)* and *Progressively Filtering* (abbreviated as *ProcPF)*. The basic idea behind SPF is to first divide the database into partitions according to the time granularity imposed. Then, in light of the exhibition period of each item, SPF employs *ProcSG* to segment the database into sub-databases in such a way that items in each sub-database will have *either* the common starting time *or* the common ending time. Note that such a segmentation will allow us of counting the itemsets in each sub-database either forward or backward (in time) efficiently. Then, for each sub-database, SPF utilizes *ProcPF* to progressively filter candidate 2-itemsets with cumulative filtering thresholds from one partition to another. Since infrequent 2-itemsets are hence filtered out in the early processed partitions, the resulting candidate 2-itemsets will be very close to the frequent 2-itemsets. This feature allows us of adopting the scan reduction technique by generating all candidate k-itemsets ($k > 2$) from candidate 2-itemsets directly [15]. The experimental results show that algorithm SPF significantly outperforms other schemes which are extended from prior methods in terms of the execution time and scalability. The advantage of SPF becomes even more prominent as the size of the database increases.

The rest of this paper is organized as follows. Section 2 describes the problem of mining general temporal association rules. The proposed algorithm SPF is presented in Section 3. The performance of algorithm SPF is empirically evaluated in Section 4. We conclude this paper with Section 5.

2. Problem Description

As described in Section 1, the items in a transaction database may have different exhibition periods. Without loss of generality, it is assumed that a certain time granularity, e.g., *week, month, quarter* or *year*, is imposed by the application database. Let n be the number of partitions divided by the time granularity imposed. In the model considered, $db^{p,q}$ $(1 \leq p \leq q \leq n)$ denotes the portion of the transaction database formed by a continuous region from the partition P_p to the partition P_q, and $X^{p,q}$ denotes the *temporal itemset* whose items are commonly exhibited from the partition P_p to the partition P_q.

As such, we can define the maximal temporal itemset $X^{p,q}$ and the corresponding temporal sub-itemsets as follows.

Definition 1: *The temporal itemset $X^{p,q}$ is called a maximal temporal itemset (TI) if P_p is the latest starting partition and P_q is the earliest ending partition of all items belonging to X. $[p, q]$ is referred to as the maximal common exhibition period (MCP) of the itemset X, denoted by $MCP(X)$.*

Definition 2: *The temporal itemset $Y^{p,q}$ is called a temporal sub-itemset (SI) of the maximal temporal itemset $X^{p,q}$ if $Y \subset X$.*

Based on Definition 1, the temporal itemset $BCD^{2,2}$ in Figure 1 is deemed a maximal temporal itemset since the latest starting partition and the earliest ending partition of the items B, C and D are both P_2. In addition, the temporal itemsets $BC^{2,2}$, $BD^{2,2}$ and $CD^{2,2}$ are the corresponding temporal sub-itemsets of $BCD^{2,2}$ according to Definition 2. Note that $BD^{2,2}$ is also a maximal temporal itemset itself, but $BC^{2,2}$ is not since the earliest ending partition of the items B and C is P_3 rather than P_2.

In the conventional problem of mining association rules, the support of the itemset X is determined by $supp(X) = \frac{|\{T \in db^{1,n} | X \subseteq T\}|}{|db^{1,n}|}$, which is referred to as the *absolute* support in this paper. However, as explained in Section 1, we shall provide an equitable support counting basis for each temporal itemset. To this end, we define the relative support for a temporal itemset below.

Definition 3: *The relative support of the temporal itemset $X^{p,q}$ is defined as*

$$supp(X^{p,q}) = \frac{|\{T \in db^{p,q} | X \subseteq T\}|}{|db^{p,q}|}$$

which is the fraction of the transactions supporting the itemset X in $db^{p,q}$.

With the equitable support counting basis defined in Definition 3, we can then determine whether a maximal temporal itemset is frequent by the following definition.

Definition 4: *The maximal temporal itemset $X^{MCP(X)}$ is termed to be frequent iff $supp(X^{MCP(X)}) \geq min\_supp$ where min_supp is the minimum support required.*

Property 1: *All temporal sub-itemsets of a frequent maximal temporal itemset are frequent.*

As will be explained later, Property 1 is very important for us to determine the confidence of a general temporal association rule defined below.

Definition 5: *The rule $(Y \Longrightarrow Z)^{MCP(X)}$ derived from the maximal temporal itemset $X^{MCP(X)}$ is called a general temporal association rule if $Y \subset X$ and $Z = X - Y$.*

Definition 6: *The confidence of the general temporal association rule $(Y \Longrightarrow Z)^{MCP(Y \cup Z)}$ is defined as*

$$conf((Y \Longrightarrow Z)^{MCP(Y \cup Z)}) = \frac{supp((Y \bigcup Z)^{MCP(Y \cup Z)})}{supp(Y^{MCP(Y \cup Z)})}.$$

Note that the calculation of the confidence of a general temporal association rule not only depends on the relative support of the corresponding maximal temporal itemset but also relys on the relative supports of the corresponding temporal sub-itemsets. Property 1 ensures that the relative supports of the corresponding temporal sub-itemsets can be obtained without extra database scans since all temporal sub-itemsets of the frequent maximal temporal itemset are also frequent.

Finally, given a pair of min_conf and min_supp required, we can define the frequent general temporal association rule below.

Definition 7: *The general temporal association rule* $(Y \implies Z)^{MCP(Y\cup Z)}$ *is termed to be frequent iff* $supp((Y \cup Z)^{MCP(Y\cup Z)}) \geq min\_supp$ *and* $conf((Y \implies Z)^{MCP(Y\cup Z)}) \geq min\_conf$.

With these definitions, the problem of mining general temporal association rules is to discover all frequent general temporal association rules from the large database. Similarly, the problem of mining general temporal association can be decomposed into two steps: (1) Generate all frequent maximal temporal itemsets (TIs) and the corresponding temporal sub-itemsets (SIs) with their relative supports; (2) Derive all frequent general temporal association rules that satisfy *min_conf* from these frequent TIs.

Note that once the frequent TIs and SIs with their supports are obtained, deriving the frequent general temporal association rules is straightforward. Therefore, in the rest of this paper we concentrate our discussion on the algorithms for mining frequent TIs and SIs.

3. Mining General Temporal Association Rules

We present the proposed algorithm, SPF, for mining general temporal association rules in this section. A detailed description of algorithm SPF is given in Section 3.1. In Section 3.2, we use an example to illustrate the operations of SPF.

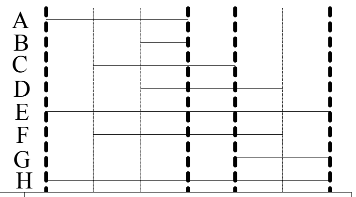
3.1. Algorithm SPF

The major challenge of mining general temporal association rules is that the exhibition periods of the items in the transaction database are allowed to be different from one to another. In such a circumstance, it is very difficult to efficiently generate candidate itemsets since the downward closure property is no longer valid as explained in Section 1. To address this problem, a novel algorithm, SPF, is proposed in this section to discover general temporal association rules efficiently.

In essence, algorithm SPF consists of two major procedures, i.e., *Segmentation (*abbreviated as *ProcSG)* and *Progressively Filtering* (abbreviated as *ProcPF)*. The basic idea behind SPF is to first divide the database into partitions according to the time granularity imposed. Then, in light of the exhibition period of each item, SPF employs *ProcSG* to segment the database into sub-databases in such a way that items in each sub-database will have *either* the common starting time *or* the common ending time. For each sub-database, SPF utilizes *ProcPF* to progressively filter candidate 2-itemsets with cumulative filtering thresholds from one partition to another. After all sub-databases are processed, SPF unions all candidate 2-itemsets generated in each sub-database. As pointed out earlier, since infrequent 2-itemsets will be filtered out in the early processed partitions, the resulting candidate 2-itemsets will be very close to the frequent 2-itemsets. This feature allows us of adopting the scan reduction technique by generating all candidate k-itemsets ($k > 2$) from candidate 2-itemsets directly. After all candidate itemsets are generated, they are transformed to TIs, and the corresponding SIs are generated based on these TIs. Finally, the frequent TIs and SIs with their supports can be obtained by scanning the whole database once.

3.1.1. Description of *ProcSG* for *Segmentation*

The motivation of *ProcSG* is to first reduce the problem of mining general temporal association rules to the one in which the exhibition periods of the items are only allowed to be *either* different in the starting time *or* different in the ending time. After such a reduction, we are able to employ *ProcPF*, in each sub-database, to progressively filter candidate 2-itemsets either forward or backward (in time) efficiently. However, as mentioned above, the advantageous feature of *ProcPF* is that it can progressively filter out infrequent 2-itemset in the early processed partitions. Thus, the more segments the whole database is divided into, the less significant the filtering effect will be. In view of this, *ProcSG* is devised to segment the whole database into the minimal number of sub-databases as required for items in each sub-database to have either the common starting partition or the common ending partition.



index	0	1	2	3	4	5	6
flag[i][L]		T	T	F	T	F	
flag[i][R]		F	F	T	T	T	
direction	-1	L	L	-1	-1	R	

Figure 2. An example illustrating the execution of *ProcSG***.**

Figure 2 illustrates the execution of *ProcSG*. Operations in lines 1-3 initialize the parameters used in the procedure. The **for** loop in lines 4-8, for each item, sets the left flag of the starting partition and the right flag of the ending partition to be $true$. The resulting values in *flag* array are shown in the bottom of Figure 2. With these flags, the **for** loop in lines 9-31 scans the partitions from the first to the last. Once the value of $direction$ is reversed, one segmentation has to be made, and the value of $direction$ is reset. From the example in Figure 2, the value of $direction$ is reversed twice, meaning that two times of segmentation are needed. Consequently, the original database $db^{1,6}$ is divided into $db^{1,3}$, $db^{4,4}$ and $db^{5,6}$.

Procedure ProcSG(n)
```
1.  $SM = \emptyset$; $direction = -1$; $head = 1$;
2.  for ($index = 0$ to $n$)
3.    $flag[i][L] = false$; $flag[i][R] = false$;
4.  for (each item $i \in \mathcal{I}$) {
5.    $(p, q) = MCP(i)$;
6.    $flag[p-1][L] = true$;
7.    $flag[q][R] = true$;
8.  }
9.  for ($i = 1$ to $n - 1$) {
10.   if ($direction == -1$) {
11.     if ($flag[i][L] == true$ and $flag[i][R] == true$) {
12.       $SM = SM \cup \{(head, index, direction)\}$;
13.       $head = index + 1$;
14.     } elseif ($flag[i][L] == true$ and $flag[i][R] == false$)
15.       $direction = L$;
16.     elseif ($flag[i][L] == false$ and $flag[i][R] == true$)
17.       $direction = R$;
18.   } elseif ($direction == L$) {
19.     if ($flag[index][R] == true$) {
20.       $SM = SM \cup \{(head, index, direction)\}$;
21.       $head = index + 1$;
22.       $direction = -1$;
23.     }
24.   } elseif ($direction == R$) {
25.     if ($flag[i][L] == true$) {
26.       $SM = SM \cup \{(head, index, direction)\}$;
27.       $head = index + 1$;
28.       $direction = -1$;
29.     }
30.   }
31. }
32. $SM = SM \cup \{(head, n, direction)\}$;
33. return $SM$;
```

3.1.2. Description of *ProcPF* for *Progressively Filtering*

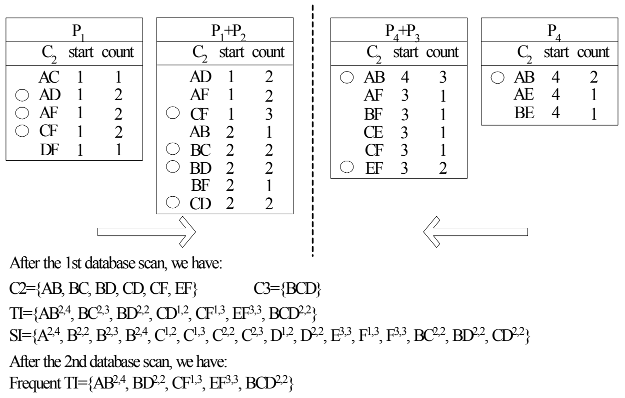
After the entire database is segmented by *ProcSG*, *ProcPF* is designed to progressively filter candidates 2-itemsets from one partition to another in each sub-database. Specifically, *ProcPF* generates all 2-itemsets and counts their occurrences in the first partition. For those 2-itemsets whose numbers of occurrences are not smaller than the filtering threshold (i.e., $min\_supp * |P_1|$), they are viewed as candidate 2-itemsets and will be brought to the next partition for further processing. Then, *ProcPF* will generate new 2-itemsets in the second partition and count their occurrences as well. However, for those candidate 2-itemsets brought from the previous partition, their numbers of occurrences will be cumulated from the previous partition to the current one. Note that the filtering threshold (i.e., $min\_supp * (|P_1| + |P_2|)$) for them will also be cumulated. Similarly, those 2-itemsets whose numbers of occurrences are not smaller than their corresponding filtering thresholds will be brought to the next partition for further processing until there is no partition to be processed any more.

Let PS denote the cumulative set of candidate 2-itemsets. It is noted that PS is composed of the following two types of candidate 2-itemsets: (1) the candidate 2-itemsets that were carried over from the previous partition and remain as candidate 2-itemsets after the current partition is included into consideration; and (2) the candidate itemsets that were not in the cumulative candidate set in the previous partition but are newly selected after taking the current partition into account. Since a significant number of 2-itemsets will be filtered out in the early partitions, the resulting PS will be very close to the set of frequent 2-itemsets after processing all partitions. Taking advantage of this feature, we can employ the scan reduction technique to generate all candidate k-itemsets where $k > 2$ from (k-1)-itemsets at the same time [15].

The procedure to progressively filter out infrequent 2-itemsets is shown in *ProcPF*. *ProcPF* takes three arguments p, q and $direction$ as the inputs, where p and q are the starting and ending partitions to be processed ($p \leq q$), and $direction$ indicates the scanning direction, i.e., either forward (i.e., from p to q) or backward (i.e., from q to p) in time. If the items in the sub-database have the same ending partition, the direction is forward. Otherwise, the direction is backward. As shown in *ProcPF*, operations in lines 1-5 initially set PS to be an empty set and determine the scanning sequence. The **for** loop, in lines 6-17, finds out all 2-itemsets with their numbers of occurrences in each partition, and employs the corresponding filtering threshold (i.e., $min\_supp * \sum_{m=X.start,h} |P^m|$) to filter out infrequent ones. After processing all partitions, the cumulative set of candidate 2-itemsets, PS, is returned in line 18.

Procedure ProcPF(p, q, $direction$)
```
1.  $PS = \emptyset$;
2.  if ($direction == left$)
3.    $head = p$; $tail = q$;
4.  else
5.    $head = q$; $tail = p$;
6.  for ($h = head$ to $tail$)
7.  for (each 2-itemset $X_2$ in $P_h$)
8.    if ($X_2 \notin PS$) {
9.      $X_2.count = N_{P_h}(X_2)$;
10.       $X_2.start = h$;
11.       if ($X_2.count \geq min\_supp * |P_h|$)
12.          $PS = PS \cup X_2$;
13.      } **else** {
14.        $X_2.count = X_2.count + N_{P_h}(X_2)$;
15.        **if** ($X_2.count < \lceil min\_supp * \sum_{m=X_2.start,h} |P_m| \rceil$)
16.          $PS = PS - X_2$;
17.      }
18.  **return** $PS$;
```

Finally, algorithm SPF is completed by the integration of *ProcSG* and *ProcPF*. At first, algorithm SPF segments the database into sub-databases by *ProcSG*. Then, for each sub-database, algorithm SPF employs *ProcPF* to progressively filter out candidate 2-itemsets. Using the scan reduction technique [15], SPF generates all candidate k-itemsets from (k-1)-itemsets. Thereafter, the candidate k-itemsets are transformed to TIs, and the corresponding SIs are generated. Finally, the database is scanned once to determine all frequent TIs and SIs.



P_1			
	C_2	start	count
	AC	1	1
○	AD	1	2
○	AF	1	2
○	CF	1	2
	DF	1	1

P_1+P_2			
	C_2	start	count
	AD	1	2
	AF	1	2
○	CF	1	3
	AB	2	1
○	BC	2	2
○	BD	2	2
	BF	2	1
○	CD	2	2

P_4+P_3			
	C_2	start	count
○	AB	4	3
	AF	3	1
	BF	3	1
	CE	3	1
	CF	3	1
○	EF	3	2

P_4			
	C_2	start	count
○	AB	4	2
	AE	4	1
	BE	4	1

After the 1st database scan, we have:

C2={AB, BC, BD, CD, CF, EF}   C3={BCD}

TI={$AB^{2,4}$, $BC^{2,3}$, $BD^{2,2}$, $CD^{1,2}$, $CF^{1,3}$, $EF^{3,3}$, $BCD^{2,2}$}

SI={$A^{2,4}$, $B^{2,2}$, $B^{2,3}$, $B^{2,4}$, $C^{1,2}$, $C^{1,3}$, $C^{2,2}$, $C^{2,3}$, $D^{1,2}$, $D^{2,2}$, $E^{3,3}$, $F^{1,3}$, $F^{3,3}$, $BC^{2,2}$, $BD^{2,2}$, $CD^{2,2}$}

After the 2nd database scan, we have:

Frequent TI={$AB^{2,4}$, $BD^{2,2}$, $CF^{1,3}$, $EF^{3,3}$, $BCD^{2,2}$}

Figure 4. Progressively Filtering candidate 2-itemsets in each sub-database.

3.2. An Illustrative Example of Algorithm SPF

The operation of algorithm SPF can be best understood by an illustrative example as shown in Figure 1. Suppose that the minimum support and confidence required are 30% and 75%, respectively, i.e., $min\_supp = 30\%$ and $min\_conf = 75\%$. As explained in Section 3.1, SPF first segments the database into several sub-databases by *ProcSG*. In our example, the transaction database, $db^{1,4}$, is segmented into two sub-databases, $db^{1,2}$ and $db^{3,4}$as shown in Figure 3. The scanning direction of $db^{1,2}$ is from the left to the right whereas the scanning direction of $db^{3,4}$ is from the right to the left.

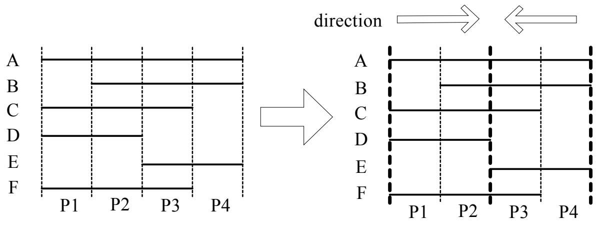


Figure 3. Segmenting the illustrative database.

After the database is segmented into sub-databases where the items in each sub-database have either the same starting or ending partition., algorithm SPF employs *ProcPF* to progressively filter the candidate 2-itemsets in each of these two sub-databases. The execution of *ProcPF* in sub-database $db^{1,2}$ is shown in the upper-left part of Figure 4. Five 2-itemsets are first considered in the partition P_1, and their numbers of occurrences and the partition in which they are first considered (i.e., P_1) are also recorded. The corresponding filtering threshold for each of them is equal to $2 = \lceil 4 * 0.3 \rceil$ since there are four transactions in P_1. As a result, only the 2-itemsets AD, AF and CF are viewed as the candidate 2-itemsets and brought to the next partition for further processing. In the partition P_2, the 2-itemsets AC, BC, BD, BF and CD are newly generated, and their numbers of occurrences and the partition in which they are first considered (i.e., P_2) are also recorded. Since AD, AF and CF are brought from the previous partition, their numbers of occurrences are cumulated. So are the corresponding filtering thresholds for them (i.e., $3 = \lceil 0.3 * (4 + 4) \rceil$). Thus, only the 2-itemsets CF, BC, BD and CD are viewed as candidate 2-itemsets after processing P_2. The execution of *ProcPF* in the sub-database $db^{3,4}$ works similarly and is shown in the upper-right part of Figure 4. Note that the scanning direction in the sub-database $db^{3,4}$ is from the right to the left, i.e., from P_4 to P_3. After scanning the sub-databases $db^{1,2}$ and $db^{3,4}$, the resulting candidate set C_2 becomes $\{AB$, BC, BD, CD, CF, $EF\}$. Using the scan reduction technique, we generate all candidate k-itemsets from candidate (k-1)-itemsets where $k > 2$. In our case, only one candidate 3-itemset BCD is generated. Then, these candidates are transformed to the TIs (i.e., $\{AB^{2,4}$, $BC^{2,3}$, $BD^{2,3}$, $CD^{1,2}$, $CF^{1,3}$, $EF^{3,3}$, $BCD^{2,2}\}$), and the corresponding SIs (i.e., $\{A^{2,4}$, $B^{2,2}$, $B^{2,3}$, $B^{2,4}$, $C^{1,2}$, $C^{1,3}$, $C^{2,2}$, $C^{2,3}$, $D^{1,2}$, $D^{2,2}$, $E^{3,3}$, $F^{1,3}$, $F^{3,3}$, $BC^{2,2}$, $BD^{2,2}$, $CD^{2,2}\}$) are generated as shown in the bottom of Figure 4. By scanning the entire database again, we can have the relative supports of all temporal itemsets in $TI \cup SI$ and then determine the frequent TIs. From the frequent TIs determined (i.e., $\{AB^{2,4}$, $BD^{2,2}$, $CF^{1,3}$, $EF^{3,3}$, $BCD^{2,2}\}$), we can thus derive all frequent general temporal association rules as shown in Example 1.2.

4. Performance Analysis

To assess the performance of algorithm SPF, we compare algorithm SPF with $Apriori^{IP}$, which is extended from algorithm Apriori to deal with the problem of mining general temporal association rules. $Apriori^{IP}$ first transforms each item to the temporal 1-itemsets with all possible exhibition periods. Then, based on the anti-monotone Apriori-like heuristic, it generates candidate temporal k-itemsets from frequent temporal (k-1)-itemsets until no candidate temporal itemset can be generated any more. The simulation program is coded in C++. The experiments are run on a computer with Pentium III CPU and 512MB RAM. As will be shown later, algorithm SPF outperforms $Apriori^{IP}$ in terms of execution time and scalability. We describe the method used to generate synthetic databases in Section 4.1. The execution time of algorithm SPF and $Apriori^{IP}$ is compared in Section 4.2. Results of scaleup experiments are presented in Section 4.3.

4.1. Generation of Synthetic Workload

The method used in this paper to generate synthetic databases is similar to the one used in [3, 15] with some modifications. In order to mimic various exhibition periods of the items in a realistic database, the modifications are made as follows. Initially, we still employ the method used in [3, 15] to generate a synthetic database. Then, we equally divide the synthetic database into P partitions to simulate the phenomenon of the time granularity required. In addition, to model the exhibition periods of the items in a realistic database, we randomly select a starting partition s in the range $[1, P]$ and the ending partition e in range $[s, P]$ for each item in the synthetic database. Finally, we scan the database once to remove the items in the transactions which are not within their exhibition periods. For example, the item A would be removed from the transaction ABC in partition 1 if the exhibition period of the item A were $[2, 4]$.

Based on such a modified method, we generate several different synthetic databases to evaluate the performance of algorithm SPF. Each of the generated database consists of $|D|$ transactions with average size of $|T|$ items. The number of different items in each database is N. The average size of the potential frequent itemsets is set to $|I|$, and the number of the potential frequent itemsets is set to $|L|$. The mean correlation level between the potential frequent itemsets is set to 0.25 in our experiments. In addition, for the simplicity of presentation, we use the notation $Tx - Iy - Dz(Nm - Ln - Po)$ to represent a database in which $|T| = x$, $|I| = y$, $D = z\ (K)$, $N = m\ (K)$, $|L| = n$ and $|P| = o$ in the following subsections.

4.2. Execution Time

In the first experiment, we investigate the execution time of algorithm SPF and $Apriori^{IP}$ by varying minimum supports. The experimental results on various synthetic databases are shown in Figure 5. As shown in Figure 5, algorithm SPF consistently outperforms $Apriori^{IP}$ in terms of the execution time. Specifically, the execution time of algorithm SPF is in orders of magnitude smaller than that of $Apriori^{IP}$. The margin even grows as the minimum support decreases.

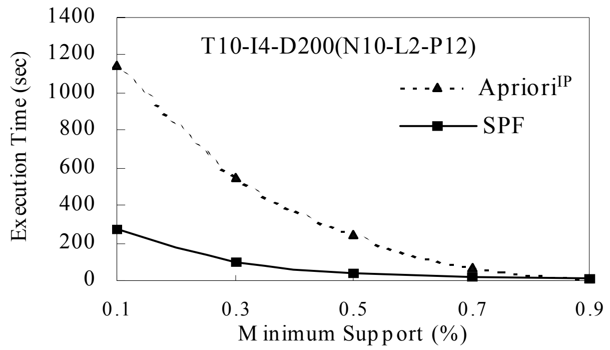


Figure 5. The execution time under various minimum supports.

The advantage of SPF over $Apriori^{IP}$ can be explained below. First, the number of candidates generated by $Apriori^{IP}$ increases exponentially as the number of items or the number of partitions increases. In contrast, the number of candidates generated by algorithm SPF is in proportion to the number of items or the number of partitions in the synthetic database. Second, $Apriori^{IP}$ needs to scan the database multiple times to determines frequent k-itemsets. However, by the technique of scan reduction, algorithm SPF only needs to scan the database twice. These two factors will be further explored in the second experiment in Section 4.3.

4.3. Scaleup Experiments

In the fourth experiment, we investigate the scalability of algorithm SPF by varying the number of transactions in the synthetic database ($|D|$). Three different minimum supports are considered in this experiment set, i.e., 0.2%, 0.4%, 0.8% respectively. The experimental result is shown in Figure 6. Note that the execution time under various numbers of transactions are normalized with respect to the time for $T10 - I4 - D100$. As shown in Figure 6, the execution time of algorithm SPF increases linearly while the number

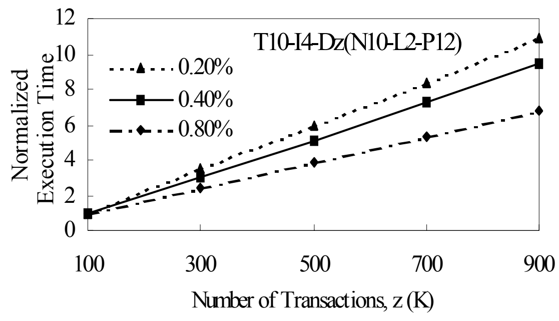


Figure 6. Normalized execution time under various numbers of transactions

of transactions in the synthetic database increases. It shows that algorithm SPF is of good scalability.

5. Conclusion

In this paper, we explored a new model of mining general temporal association rules, i.e., $(X \Rightarrow Y)^{MCP(X \cup Y)}$, from large databases where the exhibition periods of the items are allowed to be different from one to another. We developed an efficient algorithm, referred to as algorithm SPF in this paper to discover general temporal association rules effectively. The experimental results showed that algorithm SPF significantly outperforms other schemes which are extended from prior methods in terms of the execution time and scalability. With the capability of mining general temporal association rules for items with different exhibition periods, algorithm SPF outperforms prior methods in its generality and superiority.

Acknowledgment

The authors are supported in part by the National Science Council, Project No. NSC 91-2213-E-002-034 and NSC 91-2213-E-002-045, Taiwan, Republic of China.

References

[1] R. Agarwal, C. Aggarwal, and V. Prasad. A Tree Projection Algorithm for Generation of Frequent Itemsets. *Jornal of Parallel and Distributed Computing (Special Issue on High Performance Data Mining)*, 2000.

[2] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. *Proc. of ACM SIGMOD*, pages 207–216, May 1993.

[3] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules in Large Databases. *Proc. of the 20th International Conference on Very Large Data Bases*, pages 478–499, September 1994.

[4] J. Ale and G. Rossi. An Approach to Discovering Temporal Association Rules. *ACM Symposium on Applied Computing*, 2000.

[5] A. M. Ayad, N. M. El-Makky, and Y. Taha. Incremental Mining of Constrained Association Rules. *Proc. of the First SIAM Conference on Data Mining*, 2001.

[6] X. Chen and I. Petr. Discovering Temporal Association Rules: Algorithms, Language and System. *Proc. of 2000 Int. Conf. on Data Engineering*, 2000.

[7] E. C. et.al. Finding Interesting Associations without Support Pruning. *IEEE Transactions on Knowledge and Data Engineering*, pages 64–78, January/February 2001.

[8] J. Han and Y. Fu. Discovery of Multiple-Level Association Rules from Large Databases. *Proc. of the 21th International Conference on Very Large Data Bases*, pages 420–431, September 1995.

[9] J. Han, L. V. S. Lakshmanan, and R. T. Ng. Constraint-Based, Multidimensional Data Mining. *COMPUTER (special issues on Data Mining)*, pages 46–50, 1999.

[10] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. *Proc. of 2000 ACM-SIGMOD Int. Conf. on Management of Data*, pages 486–493, May 2000.

[11] C.-H. Lee, C.-R. Lin, and M.-S. Chen. Sliding-Window Filtering: An Efficient Algorithm for Incremental Mining. *Proc. of the Tenth ACM International Conference on Information and Knowledge Management*, November, 2001.

[12] J.-L. Lin and M. Dunham. Mining Association Rules: Anti-Skew Algorithms. *Proc. of 1998 Int'l Conf. on Data Engineering*, pages 486–493, 1998.

[13] B. Liu, W. Hsu, and Y. Ma. Mining Association Rules with Multiple Minimum Supports. *Proc. of 1999 Int. Conf. on Knowledge Discovery and Data Mining*, August 1999.

[14] R. Ng and J. Han. Efficient and Effective Clustering Methods for Spatial Data Mining. *Proc. of the 18th International Conference on Very Large Data Bases*, pages 144–155, September 1994.

[15] J.-S. Park, M.-S. Chen, and P. S. Yu. Using a Hash-Based Method with Transaction Trimming for Mining Association Rules. *IEEE Transactions on Knowledge and Data Engineering*, 9(5):813–825, October 1997.

[16] R. Srikant and R. Agrawal. Mining Generalized Association Rules. *Proc. of the 21th International Conference on Very Large Data Bases*, pages 407–419, September 1995.

[17] R. Srikant and R. Agrawal. Mining quantitative association rules in large relational tables. *Proc. of 1996 ACM-SIGMOD Conf. on Management of Data*, 1996.

[18] K. Wang, Y. He, and J. Han. Mining Frequent Itemsets Using Support Constraints. *Proc. of 2000 Int. Conf. on Very Large Data Bases*, September 2000.