SMCA: A General Model for Mining Asynchronous Periodic Patterns in Temporal Databases

Kuo-Yu Huang and Chia-Hui Chang, *Member*, *IEEE Computer Society*

Abstract—Mining periodic patterns in time series databases is an important data mining problem with many applications. Previous studies have considered synchronous periodic patterns where misaligned occurrences are not allowed. However, asynchronous periodic pattern mining has received less attention and only been discussed for a sequence of symbols where each time point contains one event. In this paper, we propose a more general model of asynchronous periodic patterns from a sequence of symbol sets where a time slot can contain multiple events. Three parameters min_rep, max_dis, and $global\_rep$ are employed to specify the minimum number of repetitions required for a valid segment of nondisrupted pattern occurrences, the maximum allowed disturbance between two successive valid segments, and the total repetitions required for a valid sequence. A 4-phase algorithm is devised to discover periodic patterns from a time series database presented in vertical format. The experiments demonstrate good performance and scalability with large frequent patterns.

Index Terms—Periodic pattern, asynchronous sequence, partial periodicity, temporal database.

1 Introduction

Pattern mining plays an important role in data mining tasks. Various patterns have been introduced for different applications, e.g., frequent item sets [1], [6] and sequential patterns [2], [14], [18] for transaction databases, frequent episodes [10], [11], [12] in event sequences, and frequent continuities [15], [7] for intertransaction association. Periodic patterns are recurring patterns that have temporal regularities in time-series databases. Periodic patterns exist in many kinds of data. For example, tides, planet trajectories, somite formation, daily traffic patterns, and power consumptions all present certain periodic patterns. There are many emerging applications, including stock market price movement, earthquake prediction, telecommunication network fault analysis, repeat detection in DNA sequences and occurrences of recurrent illnesses, etc.

The discovery of patterns with periodicity has been studied in several works [3], [4], [5], [9], [13]. For example, Ozden et al. proposed the mining of cyclic association rules that reoccur in every cycle of the time span of the temporal database [13]. Han et al. considered imperfect periodic patterns that reoccur for at least $minconf$ percent of the cycles [4]. Berberidis et al. further proposed an approximate periodicity detection algorithm [3]. However, these studies considered only synchronous periodic patterns and did not recognize the misaligned presence of patterns due to the intervention of random noise. For example, assume that a temporal database contains a periodic pattern, "Beer and Diaper," on Friday nights, from January to March. However, in April, the business has a big promotion for beer every Saturday. Therefore, many customers would buy beer on Saturday instead of Friday because of this promotion. In this case, it would be desirable if the pattern can still be recognized when the disturbance is within some reasonable threshold. Therefore, in [17], Yang et al. extended the idea to find asynchronous periodic patterns.

Yang et al.'s asynchronous periodic pattern problem aims at mining the longest periodic subsequence which may contain a disturbance of length up to a certain threshold. Formally, a valid subsequence with respect to a pattern P in a sequence D is a set of nonoverlapping valid segments, where a valid segment has at least min_rep contiguous matches of P and the distance between any two successive valid segments does not exceed a parameter max_dis. A valid subsequence with the most overall repetitions of P is called its longest valid subsequence. However, this model has some problems.

- First, this model only focused on mining periodic patterns in temporal sequences of events. However, in real-world applications, we may find multiple events at one time slot in terms of various intervals (e.g., hour, day, week, etc.) as discussed in previous works [4], [5], [13]. We refer to such databases as sequences of event sets.
- Second, this model only focused on mining the longest sequence of a pattern, which can only capture part of the system's behavior. For example, in the case when two successive, nonoverlapped segments with a disturbance larger than max_dis, only the larger segment will be reported. Taking another example, as shown in Fig. 1a, it is possible

• *The authors are with the Department of Computer Science and Information Engineering, National Central University, No. 300, Jungda Rd., Jhongli City, Taoyuan, Taiwan 320, R.O.C.*
E-mail: want@db.csie.ncu.edu.tw, chia@csie.ncu.edu.tw.

Manuscript received 21 May 2004; revised 14 Sept. 2004; accepted 21 Sept. 2004; published online 19 Jan. 2005.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-0147-0504.

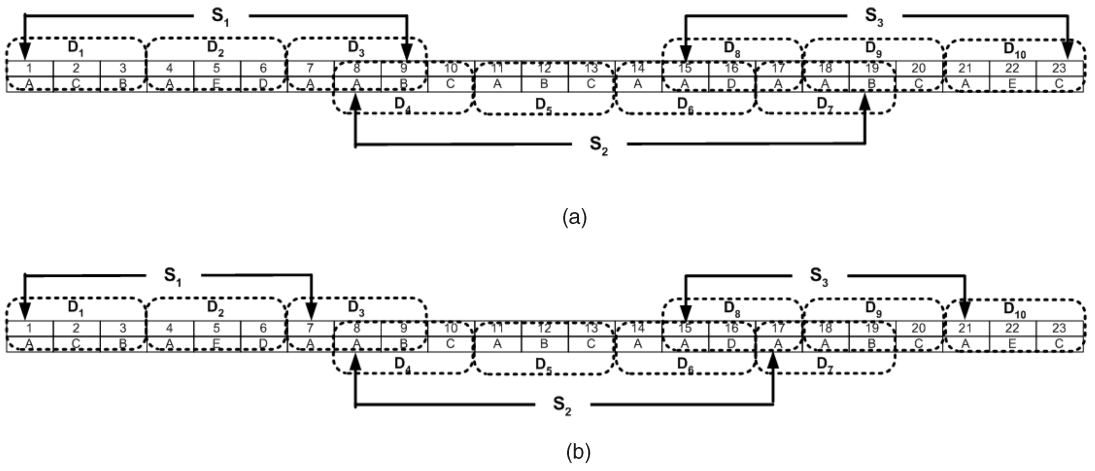

Fig. 1. Two definitions of a segment's end position: (a) Yang et al. and (b) this paper.

that two overlapped subsequences, (S_1, S_3) and (S_2), are controlled by two network administrators. Therefore, reporting the longest subsequence will miss the other subsequence.

- Third, in order to discover the longest subsequence, a longer segment can be broken into smaller segments when two segments overlap. Take Fig. 1a, for example. Suppose the min_rep is 2, the longest subsequence for pattern $(A, *, *)$ which can be connected by segments: $(D_1 \sim D_3)$, $(D_5 \sim D_6)$, and $(D_9 \sim D_{10})$ according to the definition in [17]. This, however, has to break a longer segment (both S_2 and S_3) into smaller segments to avoid overlapping. In fact, there can be several subsequences with the most overall repetitions, e.g., $(D_1 \sim D_2)$, $(D_4 \sim D_5)$, and $(D_8 \sim D_{10})$ is another subsequence with seven repetitions. We argue that a segment should be extended to its longest possibility and the overlapping of two segments should be considered as two separate sequences.
- The final problem regards the definition of a segment's end position, which affects whether or not two segments overlap. Instead of using the pattern's last occurrence as the segment's end position, the end position of a segment is defined as the period's end for the last occurrence of the pattern. To illustrate this, the end position of segment S_1 in Fig. 1a is 9 according to the definition in [17]. Therefore, segment S_1 and S_2 overlap. If we use the pattern's last occurrence as the segment's end position, as shown in Fig. 1b, the end position for segment S_1 will be 7 and segment S_1 and S_2 will not overlap. This consideration is from the application of inventory replenishment where an event occurs ahead of time as addressed in [17]. In order to connect such occurrences, the end of a segment is better defined as the event's last occurrence.

To address these problems, in this paper, we discuss asynchronous partial periodic patterns in eventset sequences where each time slot may contain multiple events. Three parameters, namely, min_rep, $global\_rep$, and max_dis are employed to qualify valid patterns and the subsequence containing them, where this subsequence, in turn, can be viewed as a list of valid segments of perfect repetitions interleaved by a disturbance. Each valid segment is required to be of maximal and at least min_rep contiguous matches of the pattern, and the distance of each piece of disturbance is allowed only up to max_dis. A sequence is termed valid if and only if the overall repetitions of the pattern are greater than $global\_rep$. We propose a series of algorithms for mining asynchronous periodic patterns, proceeding from mining valid segments for 1-patterns to mining periodic segments for i-patterns. We first devise an algorithm (called SPMiner) to discover all valid segments for each single event from temporal database presented in vertical format. The idea is to trace p possible segments for period p. When examining a time slot T_i for the given event, if T_i minus the last position for the segment maintained in $T_i\%p$ equals p, this segment is extended and its last position is updated. Then, two algorithms, MPMiner and CPMiner, are devised to discover valid segments for multievent 1-patterns and *i*-patterns. To avoid additional scans over the temporal database, we use the discovered valid segments and combine them using depth first enumeration, thus reducing redundant generation and testing. Finally, all valid segments with respect to a pattern can be combined to form an asynchronous subsequence by APMiner. In summary, this paper make the following contributions in this paper:

- A more general model of asynchronous periodic patterns is proposed to allow the mining of all patterns, not only in a sequences of events, but also in a temporal database of event sets.
- A dynamic hash-based validation mechanism is devised to discover all patterns using a single scan of the temporal database.
- There is no candidate pattern generation, as required for Apriori-like algorithms.
- Finally, the complexity analysis and experiments of the proposed algorithm show great scalability with large data sets.

The remaining parts of the paper are organized as follows: We summarize some related research in Section 2. In Section 3, we define the problem of asynchronous periodic pattern mining for temporal database. Section 4 presents our algorithm for mining asynchronous periodic patterns from temporal database and Section 5 compares the proposed algorithm with that in [17]. Experiments and performances of the algorithm study are reported in Section 6, and our conclusion are presented in Section 7.

TABLE 1
An Illustrative Temporal Database D

Time	Event Set	Time	Event Set	Time	Event Set
1	A,B,C	7	A,B,C,D	13	A,C,D
2	B,D	8	A	14	A,C
3	A,C,D	9	A,C,D	15	A,D
4	B	10	A,C	16	A,C,D
5	A,C	11	D	17	A
6	D	12	A,B,C,D	18	A,B,C,D

2 Related Work

There have been a number of recent studies in periodic pattern mining. For example, cyclic association rules proposed by Ozden et al. [13], partial periodic patterns by Han et al. [4], [5], and asynchronous periodic patterns by Yang et al. [16], [17]. However, each of these studies has a different definition for periodicity. Special terms like partial versus full, perfect versus imperfect, and synchronous versus asynchronous are used and need to be clarified.

The so called **full periodicity** specifies the behavior of the time series at all points in the period, while **partial periodicity** specifies the behavior at some but not all points in time. Partial periodicity is a looser kind of periodicity than full periodicity and its application is more general because of the mixture of periodic events and nonperiodic events in real-world data. Full periodicity search has been studied for time series of numerical values, while partial periodicity search is mainly applied for symbolic patterns. For example, Ozden et al. define the problem of discovering cyclic association rules that display regular cyclic variation over time [13]. This motivation is based on the observation that an association rule may not have the user-specified minimum confidence or support over the entire time spectrum, but its confidence and support may be above the minimum threshold within certain time intervals. In this work, a periodic pattern is expressed as $c = (p, l, o)$, where a pattern p occurs in every l time unit starting at the oth time unit ($o \leq l$). Note that what Ozden et al. considered are partial periodic patterns with **perfect** periodicity in the sense that the pattern reoccurs in every cycle, with 100 percent confidence. By studying the interaction between association rules and time, they applied three heuristics: *cycle pruning*, *cycle skipping*, and *cycle elimination* to find cyclic association rules in transactional databases.

Since real-life patterns are usually **imperfect**, Han et al. [4], [5] presented several algorithms to efficiently mine partial and imperfect periodic patterns by exploring some interesting properties related to partial periodicity, such as the Apriori property and the max-subpattern hit set property, and by shared mining of multiple periods. In order to tame the restriction cyclic association rule, Han et al. used confidence to measure how significant a periodic pattern is. The confidence of a pattern was defined as the occurrence count of the pattern over the maximum number of periods of the pattern length in the temporal database. For example, $(a, *, b)$ is a partial pattern of period 3 (The character "*" is a "don't care" character, which can match any single set of events); its occurrence count in the event series "a{b,c}baebaced" is 2; and its confidence is 2/3, where 3 is the maximum number of periods of length 3. Nevertheless, the proposed mining model works only for synchronous periodic pattern mining.

Therefore, Yang et al. [16], [17] proposed to mine for asynchronous periodic patterns that are significant using a subsequence of symbols. Two parameters, min_rep and max_dis, are employed to qualify valid patterns. The intuition is that a pattern needs to repeat itself at least a certain number (min_rep) of times to demonstrate its significance and periodicity. On the other hand, the disturbance between two valid segments has to be within some reasonable bound (max_dis). Otherwise, it would be more appropriate to treat such a disturbance as a signal of "change of system behavior" instead of random noise injected into some persistent behavior. A two-step algorithm is devised to first generate potential periods by distance-based pruning, followed by an iterative procedure to derive and validate candidate patterns and locate the longest valid subsequence for 1-patterns (called LSI). For patterns satisfying min_rep and max_dis requirements, their model will return the subsequence with the maximum overall repetitions. The second step then applies an Apriori-like algorithm (called level-wise search) to generate the subsequences of *i*-patterns based on valid subsequences of all $(i-1)$-patterns with the same period length. As argued above, this model considers only sequences of symbols, and the longest subsequences can only capture part of the system's behavior. Therefore, a more general model is proposed in this paper.

3 Problem Definition

In this section, we define the problem of asynchronous periodic mining. The problem definition is similar to [17], with some modification as discussed in Section 1. Let E be a set of all events. An event set is a nonempty subset of E. A database D is a set of time records where each time record is a tuple (tid, X) for time instant tid and event set X. A time series database stored in form of (tid, X) is called horizontal format (see Table 1). We say that an event set Y is supported by a time record (tid, X) if and only if $Y \subseteq X$. An event set with k events is called a *k*-event set.

Definition 1. *A **pattern** with period l is a nonempty sequence $P = (p_1, p_2, \ldots, p_l)$, where p_1 is an event set and others are either an event set or *, i.e., $p_j \in (2^E - \emptyset) \cup \{^*\}$ for $2 \leq j \leq l$.*

The symbol "*" is introduced to allow partial periodicity as in previous papers (the "don't care" position in a pattern). Since a pattern can start anywhere in a sequence, we only need to consider patterns that start with a non "*"

symbol. A pattern P is called an i-pattern if exactly i positions in P contain event sets. Particularly, we call 1-patterns **singular patterns**, and i-patterns **complex patterns** for $i > 1$. For example, $(A, *, *)$ is a singular pattern and (A, C, *) is a 2-pattern, which is also called complex pattern. If pattern P does not have any "*" symbol, we call it a **full pattern**. Otherwise, pattern P is called a **partial pattern**.

Definition 2. *Given a pattern $P = (p_1, p_2, \ldots, p_l)$ with period l and a sequence of l event sets $D' = (d_1, d_2, \ldots, d_l)$, we say that P **matches** D' (or D' **supports** P) if and only if, for each position j ($1 \leq j \leq l$), either $p_j = *$ or $p_j \subseteq d_j$ is true. D' is also called a **match** of P.*

In general, given a sequence of event sets and a pattern P, multiple matches of P may exist. In Fig. 1b, $D_1, D_2, \ldots, D_{10}$ are 10 matches of $(A, *, *)$. We say that two matches of the same period are overlapped if and only if they share some common subsequence, otherwise, they are disjoint. For example, D_3 and D_4 share a common subsequence at time slots 8 and 9 so they overlap, whereas D_1 and D_2 are disjoint.

Definition 3. *Given a pattern P with period l and a sequence of event sets D, a list of k ($k > 0$) disjoint matches of P in D is called a **segment** with respect to P iff it forms a contiguous subsequence of D. Here, k is referred to as the number of repetitions of this segment. For convenience, we use a 4-tuple (P, l, rep, pos) to denote a segment of pattern P with period l starting from position pos for rep times. A segment is **maximum** if there are no other contiguous matches at either end.*

In Fig. 1b, $D_4, \ldots, D_7$ are continuous and disjoint matches. Therefore, we can use $S_2 = \{(A, *, *), 3, 4, 8\}$ to indicate a segment with period 3 starting from position 8 for 4 times. For convenience, we may ignore the "don't care" symbols for 1-patterns. Therefore, segment S_2 can also be represented by $(A, 3, 4, 8)$. Note that D_4, D_5, and D_6 also form a segment but it is not maximum.

Definition 4. *A maximum segment S with respect to a pattern P is a **valid segment** if and only if the number of repetitions of S (with respect to P) is at least the required minimum repetitions (i.e., min_rep).*

Let M_1 and M_f denote the first and the last match of a maximal segment. The start (end) position of a maximal segment for a pattern is the start position of M_1 (M_f). Therefore, the start and end position of segment S_1 is 1 and 7, respectively. The disturbance between two segments is the distance between the end position of the first segment and the start position of the second segment. For Fig. 1b, the disturbance between S_1 and S_3 is 8 $(15 - 7)$.

Definition 5. *Given a temporal database D and a pattern P, a sequence in D is a set of nonoverlapping valid segments, where the distance between any two successive valid segments is less than a predefined parameter, called maximum disturbance (max_dis). A sequence is called valid if and only if the overall number of repetitions of P is greater than a predefined parameter, called global repetition ($global\_rep$).*

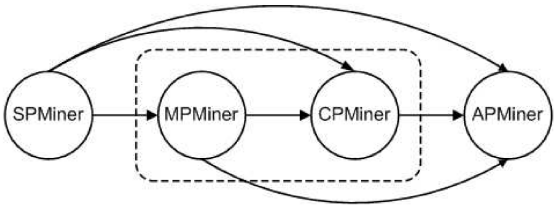

Fig. 2. The SMCA model.

For Fig. 1b, if we set $min\_rep = 2$, $global\_rep = 6$, and $max\_dis = 8$, there will be two valid subsequences (S_1, S_2) and (S_1, S_3) returned. The problem is formulated as follows: Given a temporal database and three parameters, min_rep, $global\_rep$, and max_dis, the problem is to find all valid periodic sequences with significant periods between 1 and L_{max} specified by the user.

4 Algorithm Overview

In this section, we explore methods for mining asynchronous periodic patterns in temporal database, proceeding from mining valid periodic segments for singular patterns to mining periodic segments for complex patterns. One algorithm, SPMiner, is devised to discover all valid segments for each single event from database using potential cycle detection and hash-based validation mechanism. Then, two algorithms, MPMiner and CPMiner, are devised to discover valid segments for multievent 1-patterns and complex patterns. Finally, all valid segments with respect to a pattern can be combined to form an asynchronous subsequence by APMiner.

Fig. 2 shows the architecture of our algorithm (abbreviated as SMCA). The links between the four modules show the flow of mining results. CPMiner receives the result of SPMiner and MPMiner as its input since it combines both single event 1-patterns and multievent 1-patterns to form i-patterns. The last three modules, MPMiner, CPMiner, and APMiner, are designed by depth first enumeration, which uses the mining result of previous modules as input. Note that the first three modules discover valid segments which are synchronous patterns. Therefore, their mining results can all be fed into the fourth module for asynchronous sequence mining. Finally, the rectangle around MPMiner and CPMiner indicates that the two modules can be combined in one procedure as discussed below. If the input is a sequence of symbols, MPMiner can be ignored and the three modules SPMiner, CPMiner, and APMiner can be used to discover periodic patterns for the problem defined in [17].

4.1 SPMiner: Segment Mining for Single Event Pattern

This section describes SPMiner, which discovers valid segments for each single event. In contrast to most previous research on pattern mining, which assumes a horizontal database layout, we use vertical database format which has been proved to be more efficient, as reported in [18]. Table 2

TABLE 2
Vertical Format of Database D in Fig. 1

Event	TimeList
A	1, 3, 5, 7, 8, 9, 10, 12, 13, 14, 15, 16, 17, 18
B	1, 2, 4, 7, 12, 18
C	1, 3, 5, 7, 9, 10, 12, 13, 14, 16, 18
D	2, 3, 6, 7, 9, 11, 12, 13, 15, 16, 18

shows the vertical format for database D in Table 1, where a timelist is maintained for each event. The inputs to SPMiner include a vertical format VD and the interested period interval specified by $L_{min}(=1)$ and L_{max}. By examining the variation in timelists, we devise two mining strategies, potential cycle detection (PCD) and Hash-based validation (HBV), for mining periodic segments for each single event. We explain these two techniques below and list the algorithm in Fig. 3 for reference:

- Potential Cycle Detection (PCD) is designed to recognize possible periods for an event. Since a valid pattern with period l implies there exist at least min_rep matches, we need to count the number of matches for each period l ($1 \leq l \leq L_{max}$). This can be implemented by scanning the timelist for an event once and maintaining a sliding window of L_{max} latest time instants. We use an array $CheckSet[L_{max}]$ (initialized to 1) to accumulate the counts for each period l (Step 1 in procedure PCD). At time instant T_i, if the difference between T_i and T_j, denoted by p, is less than L_{max} for T_j, $j = i-1, , i-L_{max}$, then $CheckSet[p]$ is increased by one (Steps 3-5). If, at the end, $CheckSet[l]$ is greater than min_rep, then l is a potential cycle and the HBV procedure will be executed. Take event C in Table 2 and $L_{max} = 3$ as an example. After scanning the timelist of event C, we get $CheckSet[1] = 3$, $CheckSet[2] = 8$, and $CheckSet[3] = 4$. With $min\_rep = 5$, only 2 is a possible period for event C.
- Hash-Based Validation (HBV): For each potential cycle p of an event e, this procedure scans the timelist once and outputs valid segments with period p. Note that segments represent synchronous periodic occurrences and can be overlapped as shown in Fig. 1. Therefore, a hash-based function is used to trace p independent (potentially overlapping) segments in a data structure called $CSeg[p]$, where each entry records the last position where the event occurs and the number of repetitions for the current segment. For each time instant T_i in the timelist, we compute the modulus $pos = T_i \% p$ (Step 5 in procedure HBV). The possible segment is kept in $CSeg[pos]$. If $T_i - CSseg[pos].last$ is exactly p, it implies that this event has occurred at $(T_i - p)$-th time instant. In this case, we increase $Cseg[pos].rep$ by one and update $CSeg[pos].last$ by T_i (Steps 6-7). If otherwise, $T_i - CSseg[pos].last$ is not p, it implies the last segment with period p has been interrupted. In this case, output this segment if $CSeg[pos].rep$ is greater than min_rep (Steps 8-9), and reset $CSeg[pos].rep$ to 1 and $CSeg[pos].last$ to T_i (Step 10). Finally, examine $CSeg$ once and output valid segments if the number of repetitions is greater than min_rep (Steps 12-13). Taking period 3 of event D, for example, the process of scanning $D.timelist$ is shown in Fig. 4. Initialize each record of $CSeg$ with $rep = 1$ and $last = -Max$. With $min\_rep = 5$, one valid segment $(eventset = D, p = 3, rep = 6, start = 3)$ is returned. Table 3 shows all valid segments of database D with $min\_rep = 5$ and $L_{max} = 3$.

We analyze the time complexity and space complexity of the SPMiner below. The overall time for processing

```
Procedure of SPMiner (D,L_max)
1. for each event E_i ∈ VD do
2.       PCD(E_i.TimeList);
3.       for p=1 to L_max do
4.           if ( CheckSet[p] ≥ min_rep )
5.               then HBV(E_i, E_i.TimeList, p);

Procedure of PCD(TimeList)
1. for i=1 to i <= L_max do CheckSet[i] = 1;
2. for each time instant T_i ∈ TimeList do
3.       for each time instant T_j ∈ TimeList, i < j do
4.           if (( T_j − T_i) <= L_max ) then
5.               CheckSet[T_j − T_i]++;
6.           else break;

Procedure of HBV (EvtSet, TimeList, p)
1. Allocate data structure Cseg[p];
2. for i=0 to p − 1 do /* Initialization */
3.       CSeg[i].last = −Max; CSeg[i].rep = 1;
   /* Validation */
4. for each time instant T_i ∈ TimeList do
5.       pos = T_i%p;
6.       if (T_i − Cseg[pos].last) == p) then
7.           Cseg[pos].rep ++; Cseg[pos].last = T_i; continue;
8.       if (Cseg[pos].rep ≥ min_rep) then
9.           Output (EvtSet, p, Cseg[pos].rep,
                 Cseg[pos].last − p ∗ (Cseg[pos].rep-1));
10.      Cseg[pos].rep = 1; Cseg[pos].last = T_i;
11. for i = 0 to p − 1 do /* Rechecking */
12.      if (Cseg[T_i%p].rep ≥ min_rep) then
13.          Output (EvtSet, p, Cseg[pos].rep,
                 Cseg[pos].last − p ∗ (Cseg[pos].rep-1));
```

Fig. 3. SPMiner: Singular periodic pattern mining algorithm.

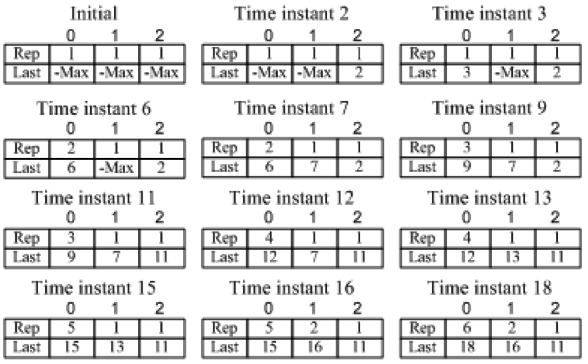

Fig. 4. Execution process for event D with period 3.

TABLE 3
Valid Segments with $min\_rep = 5$

Pattern	Period	Rep	Start
A	1	7	12
A	2	5	1
A	2	6	8
C	2	5	1
C	2	5	10
D	2	5	7
D	3	6	3

SPMiner for a given event e is $2 * n_e$ (PCD + HBV), where n_e is the number of occurrences of event e. For a given period length l, the time to find the singular periodic pattern for all events is hence $\sum_{\forall e} 2 * n_e$ which is equivalent to two database scans. Let D denote the number of time slots and T be the average number of events in each time slot. The database size can be represented by $D * T$. Consequently, the time complexity to discover all valid segments for all periods is $O(D * T * L_{max})$. The data structure used for PCD and HBV when processing an event is CheckSet and CSeg, respectively. The size of the data structure is a multiple of L_{max}, which can be reused for all events. Therefore, the space complexity is $O(L_{max})$.

4.2 Depth First Enumeration

Depth first enumeration is a popular concept used to enumerate all possible combinations. Although it has a worst-case time complexity which is exponential to the number of input elements, most enumeration stop earlier due to an antimonotone property. In this section, we will show how DFS enumeration can be used to discover valid segments for multievent singular patterns and complex patterns, and also the combination of segments with respect to one pattern to form valid sequences.

4.2.1 MPMiner for Multievent Patterns

To discover valid segments for multievent 1-patterns, we propose two mining methods: timelist-based enumeration (TBE) and segment-based enumeration (SBE). These two methods have their respective advantages and can be used in appropriate situations. We will compare these two methods in Section 6.

Timelist-Based Enumeration (TBE): For each period p, we can enumerate possible event sets from events that have valid segments with period p. Duplicate enumerates are avoided by forcing an alphabetic or numerical order on the events. For each combined event set, the timelist is obtained by the timelist intersection from the constituent events. Then, the HBV procedure used in SPMiner is applied to check if valid segments exist for the eventset. Enumeration stops whenever no valid segment exists for an event set.

Example 1. Take period 2 in Table 3, for example. There are three events A, C, D that have valid segments with period 2. We can enumerate all combinations via depth-first search. $\{A, C\}$ is first enumerated with timelist being the intersection of $A.TimeList$ and $C.TimeList$. With the timelist information, the HBV procedure is then called to check if valid segments exist for this event set $\{A, C\}$. Since valid segments exist for this event set, $\{A, C, D\}$ is then enumerated with timelist being the intersection of $AC.TimeList$ and $D.TimeList$. The process for enumerating multiple event patterns is illustrated in Fig. 5.

Segment-Based Enumeration (SBE): Another way to discover multievent 1-patterns is to combine valid segments of single-event 1-patterns. Consider segments with the same period. Let us define the normalized offset of a segment, $(EvtSet, p, rep, start)$, as the module of the segment's start position over its period, i.e., $off = start\%p$. Two overlapping segments with the same offsets can form 2-event singular patterns if the number of repetitions of the overlapped area is greater than min_rep. To discover i-event singular patterns, we can compose them from an $(i-1)$-event segment with a 1-event segment. In other words, an i-event singular pattern is composed of i segments discovered by SPMiner. For efficient combination, segments of the same period are ordered by their start position. Two segments can be combined if they have the same offsets and the overlapping area has repetitions greater than min_rep. The overlapped area is defined by the maximum start position and the minimum end position of the two

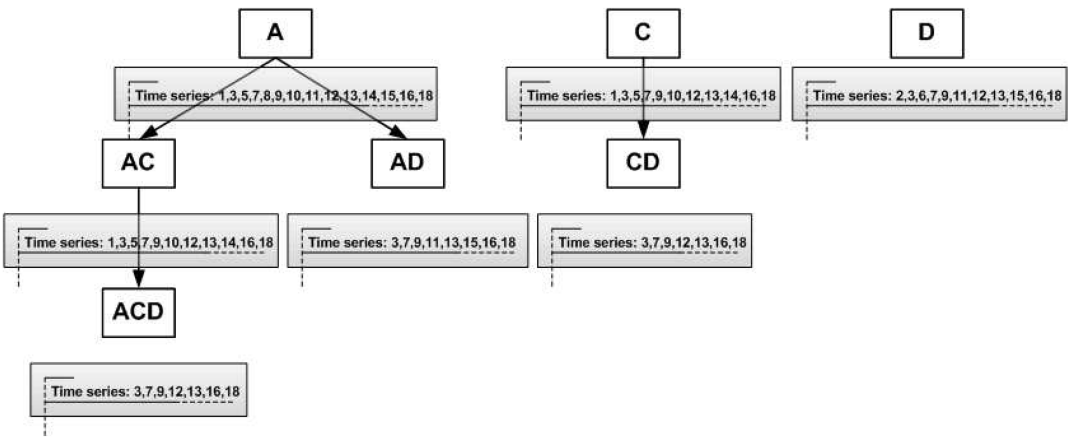

Fig. 5. Tree enumeration.

Procedure of **CPMiner**(p, $SegList_p$ **w.r.t period** p)
1. **for each segment** $S_i \in SegList_p$**; do**
2. $Node.Head$ = S_i;
3. $Node.Tail$ = all segments $S_j \in SegList$ with $j > i$;
4. $Node.start$ = $S_i.start$;
5. $Node.end$ = $S_i.start + (S_i.rep - 1) * p$;
6. CP_DFS($Node$, p);

Subprocedure of **CP_DFS**($Node, p$)
1. **if** ($|Node.Head| == p$) **then return;**
2. **for each segment** $S_i \in Node.Tail$ **do**
3. Valid = True;
4. **for each segment** $S_j \in Node.Head$ **do**
5. **if** (($S_i.start - S_j.start$)**%p==0) then**
Valid = False; break;
6. **if (Valid == False) then continue;**
7. $newC.start$ = $S_i.start$**;**
$newC.end$ = **Min**$\{Node.end, p * S_i.rep\}$**;**
8. $rep = \lceil (newC.end - newC.start)\%p \rceil$+1;
9. **if** ($rep \geq min\_rep$) **then**
10. $newC.Head$= $Node.Head \cup S_i$;
$newC.Tail$= all $S_k \in Node.Tail$ with $k > i$;
11. **PatternOutput**($newC, p, rep$)**;**
12. **CP_DFS**($newC, p$)**;**
13. **else if** ($Node.end - S_i.start$) $< p * min\_rep$) **break;**

Subprocedure of **PatternOutput**($Node$, p, rep)
1. $Shift = Node.end\%p$;
2. **for** $i = 1$ **to** p **do** $Pattern[i] = *$**;**
3. **for each segment** $S_i \in Node.Head$ **do**
4. $Pattern[(S_i.start - Shift)\%p] = S_i.EvtSet$;
5. **Output** ($Pattern$, rep, p, $Node.end - (rep - 1) * p$)**;**

Fig. 6. CPMiner: Complex periodic pattern mining algorithm

segments. Note that the end position of the segment can be determined by $start + (rep - 1)astp$. The same criteria work for combination of an $(i-1)$-event segment and a single-event segment.

Example 2. Again, take period 2 in Table 3, for example. Segment $(A, 2, 5, 1)$ and $(C, 2, 5, 1)$ can be combined to form segment $(AC, 2, 5, 1)$ since the normalized offset is the same and the overlapping area has repetition 5. However, segment $(AC, 2, 5, 1)$ and $(D, 2, 5, 7)$ cannot form a 3-event segment since the overlapped area $(min\{9, 15\} - max\{1, 7\})$ has only two repetitions.

4.2.2 CPMiner for Complex Patterns

Discovering complex patterns from singular patterns (both single-event and multievent) has a procedure similar to the segment-based enumeration (SBE) in MPMiner. We refer to this procedure as CPMiner. CPMiner enumerates possible combinations of valid segments of the same period in depth-first order and checks if a combination forms a complex pattern. For two overlapping segments with different offsets, they can form a 2-pattern if the repetition of the overlapping area is greater than min_rep. To discover an *i*-pattern, we can compose it from an $(i-1)$-pattern with 1-patterns. In other words, an *i*-pattern is composed of i segments (of 1-patterns) with different offsets. Note that two segments with the same offset can only form a singular pattern and have been considered in MPMiner.

TABLE 4
Segment List Ordered by Start Position

Seg_ID	Pattern	Period	Rep.	Start
0	C	4	7	42
1	A	4	8	46
2	B,E	4	8	49
3	D	4	6	53
4	C,F	4	9	56

For an efficient combination, segments are ordered by their start position. As shown in Fig. 6, each call of the procedure CP_DFS examines possible combinations of the current pattern composed by $Node.Head$ with a segment in the $Node.Tail$, denoted by S_i (Step 2). For a possible combination, the new segment must have a different offset with each segment in the $Node.Head$ (Steps 4-6). Two variables $Node.start$ and $Node.end$ are used to record the start and end position for the current pattern. Therefore, the overlapping area can be easily computed (Step 7). Since segments are ordered by their start position, the maximum start position is determined by the new segment. If the number of repetitions of the overlapped area is greater than the specified min_rep, the pattern is output (Steps 9-12). Since segments are ordered by their start positions, once the overlapping area is less than min_rep, the remaining segments can be ignored if the gap between $Node.end$ and $S_i.start$ is less than $p * min\_rep$ (Step 13). Another way is to check if $S_i.start$ is greater than $Node.end$, while the formula in Step 13 stop the procedure earlier.

Since a pattern $(A, *, B, C)$ can also be represented by $(B, C, A, *)$ or $(C, A, *, B)$, it is desirable to select one representation to avoid duplication. The idea is to select the one with the largest repetitions. Therefore, the first element of the pattern is determined by the segment with the minimum end position. Then, each 1-pattern, S_i is placed in the pattern with an offset determined by $(S_i.start - shift)\%p$ (Steps 4-5 of PatternOutput), where $shift$ is the offset of the segment with the minimum end position (Step 1). Note that the start position of the combined pattern will be modified by $Node.end - (rep - 1) * p$ (Step 6) in order to have the largest repetitions (Check pattern $(D, A, *, CF)$ in Table 5).

Example 3. Consider the segment list with period 4, as shown in Table 4. By using depth-first enumeration, we first enumerate $\{0, 1\}$. Since these two segments

TABLE 5
Complex Patterns Combined from Table 4

Segments	Pattern	Period	Rep.	Start
0, 2	(C, *, *, BE)	4	6	46
0, 3	(C,*,*,D)	4	5	50
1, 2	(A, *, *, BE)	4	8	46
1, 2, 4	(A, *, CF, BE)	4	6	54
1, 3	(D, A, *, *)	4	6	53
1, 3, 4	(D, A, *, CF)	4	6	53
1, 4	(A, *, CF, *)	4	6	54
2, 4	(BE, *, *, CF)	4	7	53
3, 4	(D, *, *, CF)	4	6	53

TABLE 6
Overall Comparison between SMCA and LSI

		SMCA	LSI
Single event 1-pattern	Method	SPMiner	LSI(A+B)
	Time	O($D * T * L_{max}$)	*O($k * D * T * L_{max}$)
	Space	O(L_{max})	*O($L_{max}^2 * N$)
	DB Scan	2	2
Multi-event 1-pattern	Method	MPMiner	N/A
Complex pattern (l-pattern)	Method	CPMiner	level-wise search
	DB Scan	0	l
	Candidate	No	Yes
Asynchronous subsequence	Method	APMiner	LSI(C)
	Output	All subsequences with overall rep. $\geq global\_rep$	Only the longest subsequence

have the same offset (2), they cannot be combined into a 2-pattern. Next, enumerate $\{0, 2\}$. Since they have different offsets, we then check the overlapping area with start at $Max\{42, 49\} = 49$ and end position at $Min\{66, 77\} = 66$. Since $\lceil (66 - 49)/4 \rceil + 1 = 6$ satisfies the minimum repetition 4, it forms a valid 2-pattern with segments $\{0, 2\}$. This 2-pattern is then expressed as a 4-tuple with the first element determined by segment 0 since it has the minimum end position. Then, each segment will be shifted by 2 as the offset of segment 0 is $2\ (= 42\%4)$. Therefore, segment 0 is located at $(42 - 2)\%4 = 0$ and segment 2 is located at $(49 - 2)\%4 = 3$. This will output pattern (C, *, *, BE) with six repetitions and start position determined by $66 - (rep - 1) * p = 46$. With depth-first enumeration, this 2-pattern will be combined with two other segments, 3 and 4, until no segments can be combined with the current pattern. All complex patterns discovered from Table 4 with $min\_rep = 5$ are shown in Table 5.

For an input of S_p segments with period p, there are $C_l^{S_p}$ l-patterns in the worst case. However, there are usually fewer combinations because of the min_rep constraints for the overlapping area. The correctness of CPMiner and MPMiner can be shown as follows: At each node, each element in the node's tail is combined with the node's head and regarded as a possible 1-extension. If the overlap area is less than min_rep, then we can stop any following enumeration, since any combination from that possible 1-extension would have an invalid subset (antimonotone property).

Note that it is possible to enumerate all combinations of segments discovered from SPMiner to form either multiple-event singular patterns or complex patterns. That is, the SBE approach of MPMiner and CPMiner can be combined in one depth-first enumeration, where the offset criteria is lifted and only the overlap criteria is enforced.

4.2.3 APMiner for Asynchronous Pattern

As noted in Definition 5, an asynchronous periodic pattern is defined by the existence of a valid sequence which is a set of nonoverlapping valid segments with respect to a pattern. Therefore, a depth-first algorithm is designed to enumerate all combinations of segments with respect to a pattern. Suppose segments are ordered by their start position. A single segment is itself a subsequence and potentially valid if the number of repetitions is greater than $global\_rep$. For each enumeration, we try to extend the current subsequence by examining one more segment. Then, if the start position of the new segment is within max_dis of the current subsequence, the subsequence is extended. Once the start position of a segment is greater than the end position of the current sequence by max_dis, the remaining segments can be ignored since segments are ordered by their start position.

5 Discussion

In this section, we compare the mining procedure of our proposed algorithm, SMCA, with the LSI algorithm proposed in [17]. Furthermore, we discuss the solution when the data is too long/large to fit in memory space.

5.1 Comparison

The comparison of LSI and SMCA is summarized in Table 6, where LSI is decomposed for comparison with the four modules in SMCA. Since the first step of LSI is used to discover asynchronous subsequences for 1-patterns, it is roughly equivalent to SPMiner+APMiner. To be more specific, the discovery process of LSI's first step moves among three phases for segment validation (phase A), segment growth (phase B), and sequence extension (phase C). Therefore, phase A+B is equivalent to SPMiner, and phase C is equivalent to APMiner. The second step of LSI is then used to compute asynchronous subsequences for complex patterns, which is roughly equivalent to CPMiner. Finally, since LSI is designed for sequences of events, there is no corresponding procedure for MPMiner.

The time complexity to discover the "longest" single event subsequence from a database of events is $k * M * L_{max}$, where L_{max} is the maximum period length, M is the database size, and k is abbreviated for $min\_rep + max\_dis + L_{max}$ [17]. For a database of events, the size of the database can be represented by $D * T$ for D time slots, each with an average of T events ($M = D * T$). The space complexity of LSI is

$$(max\_dis + L_{max}) * N * L_{max} + min(N * L_{max}, min\_rep * L_{max}^2 * N)$$

as analyzed in [17]. To discover valid segments according to the definition in this paper, the space of LSI can be approximated by $O(L_{max}^2 * N)$. The LSI algorithm needs two database scans, one for the distance-based pruning procedure and the other for the 1-pattern verification procedure. Analysis of SPMiner was made in the previous section. Compared to SPMiner, the vertical format representation has significant benefits over horizontal format.

Yang et al. employs a level-wise search algorithm (a priori-like) for discovering complex patterns, i.e., it generates candidate l-patterns based on $(l-1)$-patterns with the same period length and then applies LSI to validate the longest subsequence for the l-pattern. However, a priori-like algorithms often perform as many passes over the database as the length of the longest l-pattern. In this case, the number of database scans is proportional to the number of candidates. If there are $\hat{N}_p$ valid patterns with period p, this will imply a presence of $O(C_l^{\hat{N}_p}(p-l))$ candidate patterns, each of which is explicitly examined by their algorithm. Therefore, CPMiner may outperform LSI when the number of segments is not exceptionally high.

Finally, the LSI algorithm outputs the longest subsequence for a pattern. As stated in the introduction, there can be several subsequences with the most overall repetitions. Which subsequence will be output depends on the implemented algorithm. Therefore, APMiner is designed to output all valid subsequences with overall repetitions greater than $global\_rep$, as described in the problem definition.

5.2 Extra Long/Large Sequence

Sometimes, the whole time series database is too long/large to fit in memory space. In this case, we mine periodic patterns by a partition-and-validation strategy. First, the algorithm subdivides the extra-large sequence data into n nonoverlapping horizontal partitions. Each partition can be handled in memory by SPMiner. Further, each partition is transformed into vertical format. For the first partition, all valid segments can be mined by the initialization and validation steps in procedure HBV. For the succeeding partitions, the initial start (last) position is inherited from the last partition. In the final partition, valid segments are discovered by the complete three steps of procedure HBV, that is, rechecking is only necessary for the last partition. For example, suppose the memory only maintains six time records at one time. Thus, Table 1 will be divided into three partitions. Take event D for an example, the timelist in three partition will be $\{2,3,6\}$, $\{7,9,11,12\}$, and $\{13,15,16,18\}$, respectively. For the first partition, the mining process is the same, as shown in Fig. 4, from initial to time instant 6. For the second partition, the initial state is exactly that of time instant 6 and the mining process goes similarly as shown in the same figure from time instant 7 to time instant 12. Finally at partition 3, we recheck the CSeg structure after processing time slot 18 and output segment (D, period = 3, rep = 6, start = 3).

TABLE 7
Meanings of Symbols

Sym	Definition	Default
$\|D\|$	# of time instants	50K
$\|N\|$	# of events	1K
T	# of events in a time instant	10
$\|C\|$	# of potential complex patterns	5
L	# of singular pattern for each potential pattern	4
I	# of events for singular patterns	4
P	Average period length	20
S	# of segments for each potential pattern	10
Rep	Average repetition of a segment	25
Dis	Average length of disturbance	50

6 Experiments

In this section, we report on a performance study of the algorithms proposed in this paper and an application of periodic pattern mining in a real data set. We first investigate the performance of SPMiner for singular patterns and compare the result with the LSI (Longest Sequence Identifier) algorithm proposed in [17] using synthetic data. We also compare two multievent pattern mining algorithms, TBE and SBE. Then, complex pattern mining using CPMiner is also evaluated. Finally, the algorithms are applied to real data for periodic pattern mining.

6.1 Synthetic Data

For the purpose of performance evaluation, we use synthetically generated temporal data set consisting of $|N|$ distinct symbols and $|D|$ time instants. A set of periodic complex patterns C, is generated as follows: First, we decide the period length from normal distribution with average length P. Then, L $(1 < L < P)$ positions are chosen for nonempty event sets. The average number of events for each singular pattern is set to I. The number of repetitions of a segment follows a geometrical distribution with mean Rep. Following each segment, a disturbance is given, based on a geometrical distribution with mean Dis. This process repeats until S segments are generated. A total of $|C|$ complex patterns are generated. With all periodic patterns generated, we then assign events to each time instant. The number of events in each time instant is picked from a Poisson distribution with mean T. For each time instant, if the number of the events in this time instant is less than T, the insufficient events are picked at randomly from the symbol set N. Table 7 shows the notations used and their default values. The experiments are conducted on a computer with a CPU clock rate of 1GHz and 1.5GB of main memory, and the program is written in Visual C on a Windows 2000 platform.

Before presenting the experimental results, we first describe our implementation of LSI. Since the LSI algorithm is designed to discover the longest subsequence for single-event singular patterns from a sequence of events instead of event set, we made two modifications. First, we implement only the first two phases of LSI for valid segment discovery, since the last phase is used to identify the longest subsequence (The discovery process of LSI's first step

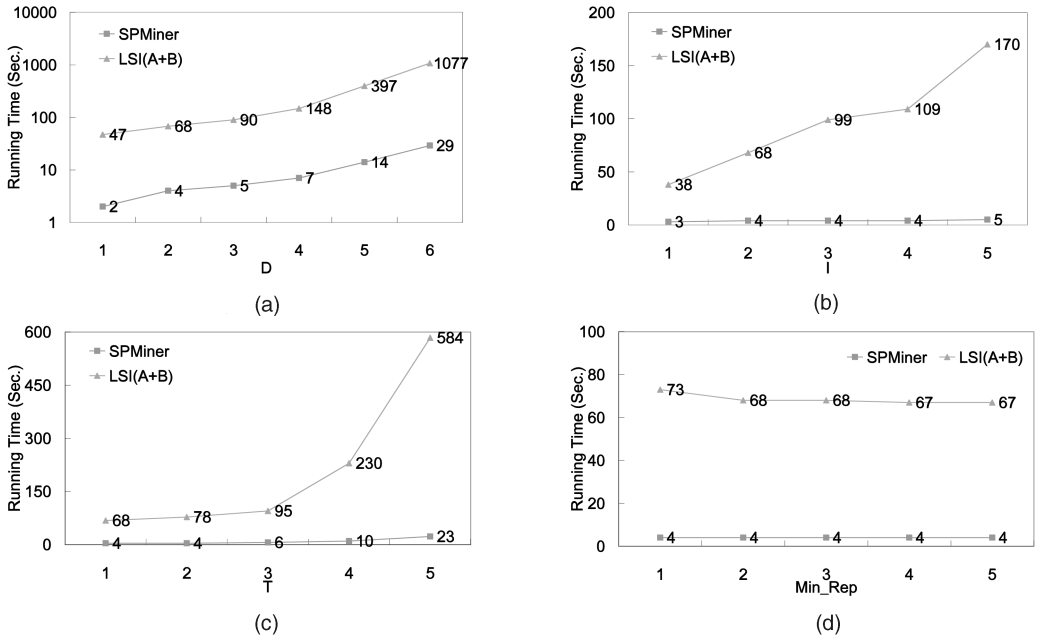

Fig. 7. Performance comparison SPMiner versus LSI.

moves among three phases for segment validation (phase A), segment growth (phase B), and sequence extension (phase C)). Second, in order to handle an eventset sequence, we implement LSI using vertical data formats to save extra time when examining a time slot in horizontal data formats.

We start by looking at the performance of SPMiner with default parameter $min\_rep = 15$ and $L_{max} = 20$ (Default Data Set). The scalability of SPMiner is shown in Fig. 7a. The scaling with database size was linear, and the running time for SPMiner was also better than LSI(A+B) (by a magnitude of 30 for $|D| = 150K$). Fig. 7b shows the effect of parameter I on LSI. When the number of events for singular patterns increases, LSI requires more time to process the increased number of candidates. As for SPMiner, the running time is independent of I since the time complexity is related only to data size D and T (note that I is a value smaller than T). In Fig. 7c, the total running time for SPMiner is linear to the average transaction size, as analyzed in Section 4.1; whereas, the running time for LSI increases dramatically since the distance-based pruning technique has comparatively less to prune. Another experiment, recorded in Fig. 7d, shows that the total running time of LSI is decreasing along with the increasing of the min_rep since the number of valid segments decreases as min_rep are increased, whereas SPMiner has minor variations since the number of repetitions is in a geometrical distribution.

Next, we demonstrate the efficiency of MPMiner by comparing timelist-based enumeration (TBE) with segment-based enumeration (SBE). Fig. 8a shows the execution time of these two methods. As we can see, SBE outperforms TBE when I and S are small. However, when I and S are large, TBE outperforms SBE. This is because the time complexity of SBE has an exponential relation to the number of segments in the worst case. In contrast, TBE is comparably stable with respect to the number of segments.

Finally, we show the performance for CPMiner. As analyzed in Section 4.2.2, the time complexity for CPMiner is related to the number of segments. Hence, we use various values of L, the maximum lengths of complex patterns, to generate different data sets. Note that for various lengths of complex patterns (L), the number of segments generated is proportional to $2^I * 2^L * C * S$. As shown in Fig. 8b, the

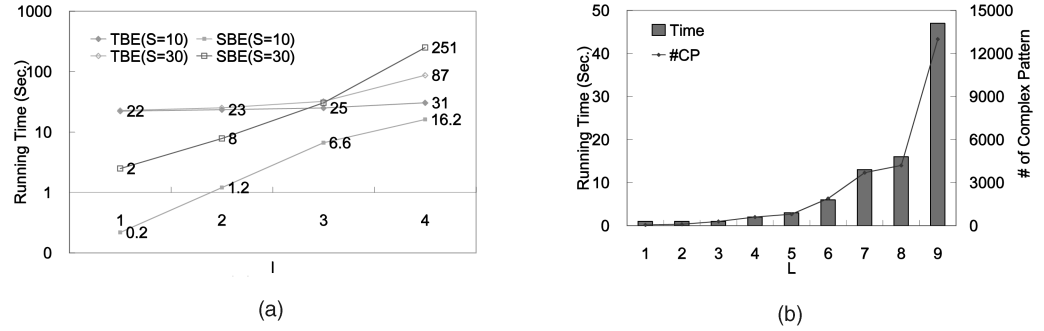

Fig. 8. Performance of (a) MPMiner (TBE versus SBE) and (b) CPMiner with parameter C = 1.

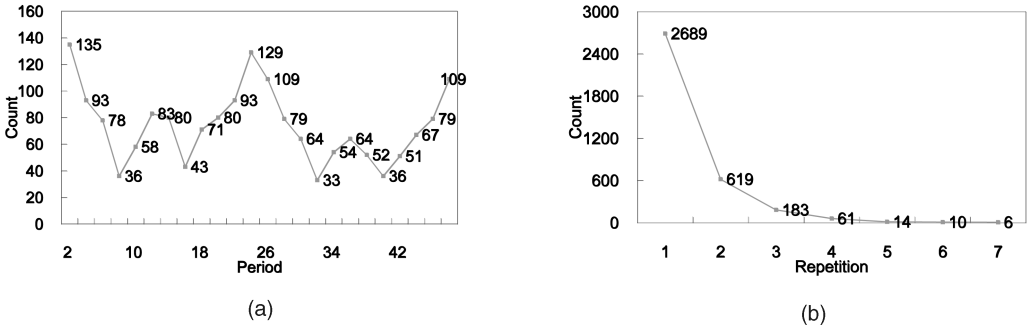

Fig. 9. KMIP Web Log Analysis. (a) Number of segments versus period. (b) Number of segments versus repetition.

execution time and the number of complex patterns have very similar trends, indicating that the running time is proportional to the number of segments. This seems to prove our estimation since nonoverlapping segments can be ignored due to the increasing order of the start position for segments.

6.2 Web Log

We also apply our algorithm to part of the Web log data for the Kinmen Information Portal Web site (http://kmip.org/), collected from 20 August 2002 to 4 June 2003. There are 6,892 time instants (hours) and 488 users in the KMIP-Log with anonymous users removed. The execution time is less than 1 sec. In average, there are four distinct users online in every hour. The interested period is a range between 1 to 48 hours. With $min\_rep = 3$, Fig. 9a shows the number of valid segments for different periods. This curve, with an acme located on 24, demonstrates that most of the users of KMIP have a regular log-in periodicity of 24 hours, reflecting a daily schedule. This curve is useful when the period of the database is unknown. Analysts can find all singular periodic patterns within a predefined period interval to find an acme of the curve. With such an acme, a possible periodicity is consequently ascertained. Fig. 9b shows that the number of valid segments decreases steeply with the increasing min_rep threshold following a geometrical distribution as our synthetic data.

6.3 Biological Data

We also apply SMCA to discover periodic conservation of the protein sequences, which is an important problem in bioinformatics. We used data in the PROSITE database of the ExPASy Molecular Biology Server (http://www.expasy.org/). We selected a protein sequence P17437 (Skin secretory protein XP2) with a known periodic pattern $\{A, P, A, P, A, *, *, E, *, *\}$, as reported in [8]. As expected, several periodic patterns which are related to the known periodic conservation are discovered. It is indicated that our algorithm can be used in protein sequence. It is worth to note that we also discover an interesting and longest pattern $\{A, P, A, P, A, E, G, E, A, P\}$ occur 11 times (approximately 46 percent) in the known periodic pattern. It may be a core pattern, since the partial slots of the pattern allow some mutations.

7 Conclusion

In this paper, a general model for asynchronous periodic pattern mining is defined. A four-phase algorithm which includes singular periodic pattern mining (SPMiner), multievent periodic pattern mining (MPMiner), complex periodic pattern mining (CPMiner), and asynchronous sequence mining (APMiner) is devised to solve the problem. One of the main contributions of this paper is that SMCA is devised to discover all patterns via two scans of the temporal database. Transforming horizontal database into their vertical formats can be done in a straightforward way. Also, SMCA can be modified for incremental mining with proper management of the mining results (both final and intermediate). Maintaining a maximal patterns can also save additional time for unnecessary mining. More works will be reported in the near future.

Acknowledgments

This work was sponsored by the Ministry of Economic Affairs, Taiwan under grant 93-EC-17-A-02-S1-029.

References

[1] R. Agrawal and R. Srikant, "Fast Algorithms for Mining Association Rules," *Proc. 20th Int'l Conf. Very Large Data Bases (VLDB '94)*, pp. 487-499, 1994.

[2] R. Agrawal and R. Srikant, "Mining Sequential Patterns," *Proc. 11th Int'l Conf. Data Eng. (ICDE '95)*, pp. 3-14, 1995.

[3] C. Berberidis, I. Vlahavas, W. Aref, M. Atallah, and A. Elmagarmid, "The Discovery of Weak Periodicities in Large Time Series," *Proc. European Conf. Principles and Practice of Knowledge Discovery in Databases (PKDD '02)*, 2002.

[4] J. Han, G. Dong, and Y. Yin, "Efficient Mining Paritial Periodic Patterns in Time Series Database," *Proc. 15th Int'l Conf. Data Eng. (ICDE '99)*, pp. 106-115, 1999.

[5] J. Han, W. Gong, and Y. Yin, "Mining Segment-Wise Periodic Patterns in Time-Related Databases," *Proc. Fourth ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '98)*, pp. 214-218, 1998.

[6] J. Han, J. Pei, and Y. Yin, "Mining Frequent Patterns Without Candidate Generation," *Proc. ACM SIGMOD Int'l Conf. Management of Data (SIGMOD '00)*, pp. 1-12, 2000.

[7] K.Y. Huang and C.H. Chang, "Prowl: An Efficient Frequent Continuity Mining Algorithm on Event Sequences," *Proc. Sixth Int'l Conf. Data Warehousing and Knowledge Discovery (DaWaK '04)*, pp. 351-360, 2004.

[8] M.V. Katti, R. Sami-Subbu, P.K. Ranjekar, and V.S. Gupta, "Amino Acid Repeat Patterns in Protein Sequences: Their Diversity and Structural-Function Implications," *Protein Science*, no. 9, pp. 1203-1209, 2000.

[9] S. Ma and J. Hellerstein, "Mining Partially Periodic Event Patterns with Unknown Periods," *Proc. Int'l Conf. Data Eng. (ICDE '01),* pp. 205-214, 2001.
[10] H. Mannila, H. Toivonen, and A.I. Verkamo, "Discovering Frequent Episodes in Sequences," *Proc. First Int'l Conf. Knowledge Discovery and Data Mining,* pp. 210-215, 1995.
[11] H. Mannila, H. Toivonen, and A.I. Verkamo, "Discovering Generalized Episodes Using Minimal Occurrences," *Proc. Second Int'l Conf. Knowledge Discovery and Data Mining,* pp. 146-151, 1996.
[12] H. Mannila, H. Toivonen, and A.I. Verkamo, "Discovering Frequent Episodes in Event Sequences," *Data Mining and Knowledge Discovery,* vol. 1, no. 3, pp. 259-289, 1997.
[13] B. Ozden, S. Ramaswamy, and A. Silberschatz, "Cyclic Association Rules," *Proc. 14th Int'l Conf. Data Eng. (ICDE '98),* pp. 412-421, 1998.
[14] R. Srikant and R. Agrawal, "Mining Sequential Patterns: Generalizations and Performance Improvements," *Proc. Fifth Int'l Conf. Extending Database Technology (EDBT '96),* pp. 3-17, 1996.
[15] A.K.H. Tung, H. Lu, J. Han, and L. Feng, "Efficient Mining of Intertransaction Association Rules," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 1, pp. 43-56, 2003.
[16] W. Wang, J. Yang, and P.S. Yu, "Mining Patterns in Long Sequential Data with Noise," *Proc. ACM SIGKDD Int'l Conf. Knowledge Discovery and Data Mining (KDD '00),* pp. 28-33, 2000.
[17] J. Yang, W. Wang, and P.S. Yu, "Mining Asynchronous Periodic Patterns in Time Series Data," *IEEE Trans. Knowledge and Data Eng.,* vol. 15, no. 3, pp. 613-628, 2003.
[18] M.J. Zaki, "Spade: An Efficient Algorithm for Mining Frequent Sequences," *Machine Learning,* vol. 42, nos. 1/2, pp. 31-60, 2001.

Kuo-Yu Huang is currently a PhD student of computer science information engineering at National Central University, Taiwan, ROC. He is also a member of the Database Laboratory. His research interests include the design of efficient, scalable, and flexible algorithms for various data mining techniques such as intratransaction rule, intertransaction rule, and temporal data mining. He has published more than 10 books on computer programming language in Chinese.

Chia-Hui Chang received the BS degree in computer science and information engineering from National Taiwan University, Taiwan, in 1993 and received the PhD degree in the same department in January 1999. She is an assistant professor in the Department of Computer Science and Information Engineering, National Central University in Taiwan. She worked as a postdoctoral in Chun-Nan Hsu's group after graduation, then joined National Central University from August 1999. Her research interests include information retrieval, knowledge discovery from databases, machine learning, and Web related research. She is a member of the IEEE Computer Society.