Proceedings of the IASTED International Conference
DATABASES AND APPLICATIONS
February 17-19, 2004, Innsbruck, Austria

ASYNCHRONOUS PERIODIC PATTERNS MINING IN TEMPORAL DATABASES

Kuo-Yu Huang and Chia-Hui Chang
Department of Computer Science and Information Engineering,
National Central University, Chung-Li, Taiwan 320
email: *want@db.csie.ncu.edu.tw*, *chia@csie.ncu.edu.tw*

ABSTRACT
Mining periodic patterns in temporal database 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 was 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 non-disrupted pattern occurrences, the maximum allowed disturbance between two successive valid segments, and the total repetitions required for a valid sequence. A four-phase algorithm is devised to discover periodic patterns from a temporal database presented in vertical format. The experiments demonstrate good performance and scalability with large frequent patterns.

KEY WORDS
Temporal Databases, Data Mining, Knowledge Discovery, Applications

1 Introduction

Pattern mining plays an important role in data mining tasks, e.g. frequent pattern, sequential pattern[4], inter-transaction pattern[5] and episode mining[2], etc. Periodic pattern mining is the problem that regards temporal regularity. For example transactional data, we may find that a pattern, Beer and Diaper, occurs at every Friday night for 20 weeks continuously. This is what we call a cyclic association rule which can be applied in period predictions such as stock data, web logs, weather data, as well as earthquake and sales records.

The discovery of pattern with periodicity has been studied in [1, 3]. However, these studies considered only synchronous periodic patterns but did not recognize the misaligned presence of patterns due to the intervention of random noise. Therefore, in [6] Yang et al. extended the periodic pattern by introducing a concept from information theory to address noisy symbols. For example, assume that a temporal database has a periodic pattern, “Beer and Diaper”, on Friday night, 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. Therefore, it would be desirable if the pattern can still be recognized when the disturbance is within some reasonable threshold.

Yang’s asynchronous periodic pattern problem aims at mining the longest periodic subsequence which may contain a disturbance of length up to a certain threshold. However, the model they built has some drawbacks. First, they only focused on periodic patterns with single event. However, in transaction databases, we may find multiple events at one time slot.We call such databases multi-event temporal database. Second, they only focused on mining the longest sequence of a pattern. In order to discover the longest subsequence, a longer segment can be broken into small segments when two segments overlap. Nonetheless, the overall repetition of the sequence is not increased than maintaining the longer segments. We argue that a segment should be extended to its longest possibility and overlapping of two segments should be considered as two separate sequences. In this case, the longest subsequence is connected by S_1 and S_3, and sequence S_2 is simply ignored. However, S_2 can be another administrator’s behavior. In other words, discovering the longest subsequence can only capture part of the system’s behavior.

In this paper, we discuss asynchronous partial periodic patterns in multi-event temporal database. Three parameters, namely min_rep, $global\_rep$ and max_dis are employed to qualify valid patterns and the subsequence containing them, where this subsequent 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 at least min_rep contiguous repetitions of the pattern and the distance of each piece of disturbance is allowed only up to max_dis. The overall number of repetitions of a sequence is equal to the sum of the repetitions of its valid segments. A sequence is termed valid if and only if the overall repetitions of the pattern are greater than $global\_rep$. We propose a four-phase algorithm for mining asynchronous periodic pattern. We first introduce a hash-based validation mechanism to discover all single event periodic patterns, named SPMiner (Single event pattern validation). In order to generate the multi-event periodic pattern, complex pattern and asynchronous sequences, we employ depth first enumeration approach to develop MP-

Miner (Multiple event pattern validation), CPMiner (Complex pattern validation) and APMiner (Asynchronous pattern validation). In summary, we have 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 database of events, but also in a database of eventsets.
- A valid segment can be represented in a compressive representation by its pattern, period, repetition and start position.
- A dynamic hash-based validation mechanism is devised to discover all singular patterns using twice scan of the temporal database.
- There is no candidate pattern generation, as required for an Apriori-like algorithm in complex pattern generation.
- We also analyze the time and space complexity and prove the correctness of the proposed algorithm.

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. Experiments and performances of the algorithm study are reported in Section 5. Finally, we conclude our study in Section 6.

2 Related Work

There have been a number of recent studies in periodic pattern mining. Ozden et al.[3] defined the problem of discovering cyclic association rules as finding cyclic relationships between the presence of items within transactions. In their research, the input data was a set of transactions, in which each transaction consisted of a set of items. In addition, each transaction was tagged with an execution time. 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.

Han et al.[1] presented several algorithms to efficiently mine partial 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 (“*” 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. [6] proposed the model to mine asynchronous periodic patterns that are significant using a subsequence of symbols which may contain a disturbance of length up to certain threshold. They propose three strategies: distanced-based pruning, single pattern verification and complex pattern verification. The discovery process of single pattern verification contains three phase: segment validation (phase A), valid segment growth (phase B) and sequence extension (phase C). For patterns satisfying min_rep and max_dis requirements, their model will return the subsequence with the maximum overall repetitions. As argued above, this model considers only sequences of symbols, and the longest subsequences can only capture part of the system behavior. In addition to the overlapped segments of two separate administrators described above, non-overlapped segments can form individual sequences due to long disturbance. Mining the longest subsequence can not present such differences. 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 [6], except for the multi-event sequence input and sequence formulation. Let E be a set of all events. An event set is a non-empty subset of E. A temporal 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 multi-event temporal database stored in form of (tid, X) is called horizontal format. We say that an event set Y is supported by a time record (tid, X) iff $Y \subseteq X$. An event set with k events is called a k-event set.

Definition 3.1 *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 periodicy 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-pattern **(singular pattern)**, and $i-$pattern **(complex pattern)** for $i > 1$. For example, $(A, *, *)$ is a singular pattern; (A, C, *) is a 2-pattern which is also called complex pattern. If pattern P doesn’t have any ”*” symbol, we call it a **full pattern**. Otherwise pattern P is called a **partial pattern**.

Definition 3.2 *Given a pattern $P = (p_1, p_2, \ldots, p_l)$ with period l and a sequence of l event sets $D' =$*

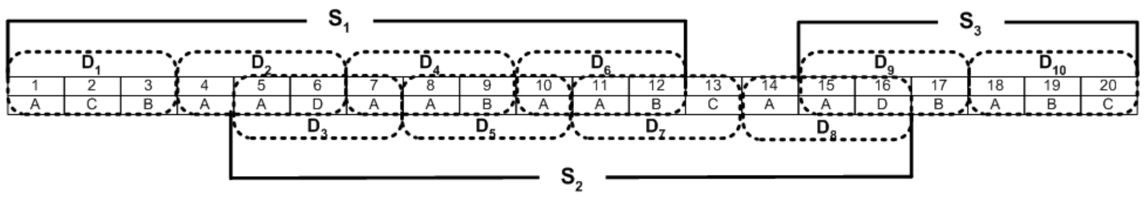


Figure 1. An example of periodic pattern

$(d_1, d_2, \ldots, d_l)$*, we say that* P ***matches*** D' *(or* D' ***supports*** P*) if an 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 Figure1, $D_1, D_2, \ldots, D_{10}$ are ten 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_2 and D_3 share a common subsequence at time slots 5 and 6 so they overlap whereas D_1 and D_2 are disjoint.

Definition 3.3 *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 *if and only if it forms a contiguous subsequence of* D*. Here,* k *is referred to as the number of repetitions of this segment.*

Definition 3.4 *A segment is* ***maximum*** *if there is no other contiguous matches at both ends. Note that the end of a segment is defined as the position of the last occurrence for non-"*" event set in* P*. Two segments are overlapped if they share common subsequence.*

In Figure1, D_1, D_2, D_4 and D_6 are continuous and disjoint matches. Therefore, we can use $S_1 = \{(A, *, *), 3, 4, 1\}$ to indicate a segment with period 3 start from position 1 for 4 times. Note that D_1, D_2 and D_4 also form a segment but it is not maximum. The end position of segment S_1 is 10. Segment S_1 and S_2 are overlapped, while segment S_2 and S_3 are not overlapped.

Definition 3.5 *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 of M_1 (M_f). Therefore, the start and end position of segment S_1 are 1 and 10, 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 Figure 1, the disturbance between S_1 and S_3 is 5 $(15 - 10)$.

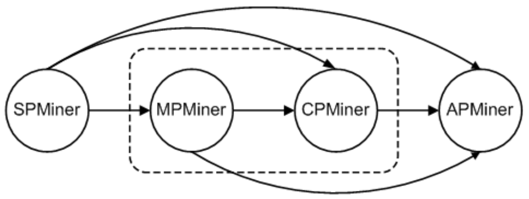


Figure 2. SMCA model

Definition 3.6 *Given a multi-event database* D *and a pattern* P*, a sequence in* D *is a set of non-overlapping valid segments, where the distance between any two successive valid segments is less than a predefined parameter, called maximum disturbance (*max_dis*). The overall number of repetitions of a sequence is equal to the sum of the repetitions of its valid segments. 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 (*$global\_rep$*).*

For Figure1, if we set min_rep = 2, $global\_rep$ = 6 and max_dis = 6, there are two valid sequence (S_1, S_3) and (S_2, 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 patterns 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 multi-event 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 temporal database in vertical format. Then, two other algorithms, MPMiner and CPMiner are devised to discover valid segments for multi-event 1-pattern and complex patterns. Finally, all valid segments with respect to a pattern can be combined to form an asynchronous sequence by APMiner.

Figure 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

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

Figure 3. Vertical format of temporal database D

both single event 1-patterns and multi-event 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 module 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 rectangular block outside 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 [6].

4.1 SPMiner: Segment Mining for Single Event Pattern

In contrast to most previous research on pattern mining, which assumes a horizontal database layout, we use vertical database format. Figure 3 shows the vertical format for database VD, where a timelist is maintained for each event. By examining the variation of timelists, we devise two mining strategies for mining periodic segments of single events.

- Potential Cycle Detection (PCD): According to the Definition 3.5, a valid pattern with period l valid implies there exist at least min_rep matches. Therefore, we first use an array $CheckSet[l]$ to accumulate the counts for each period l ($1 \leq l \leq L_{max}$). If the $CheckSet[l]$ is greater than min_rep, it is a potential cycle. Take event D in Figure 3 for 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. This can be implemented by scanning the timelist for an event once and maintaining a sliding window of L_{max} latest time instants. At time instant T_i, if the difference between T_i and T_j, denoted by p, is less than L_{max} for time instant T_j, $j = i-1, \ldots, i-L_{max}$, then $CheckSet[p]$ is increased by one.

- 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 Figure 1. This is implemented by keeping tracks of p independent (potentially overlaping) segments in a data structure called $CSeg$, where each $CSeg$ records the last position where the event occurs and the number of repetitions for current segment. For each time instant T_i in the timelist of an event, we compute the modulus $pos = T_i\%p$. 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. If otherwise, $T_i - CSseg[pos].last$ is not p, it implies the last segment has been interrupted. In this case, output this segment if $CSeg[pos].rep$ is greater than min_rep and reset $CSeg[pos].rep$ to 1 and $CSeg[pos].last$ to T_i. Finally, examine $CSeg$ once and output valid segments if the repetitions are greater than min_rep. Taking period 3 of event D for example, the process of scanning $D.timelist$ is shown in Figure 4. Initialize each record of $CSeg$ with $rep = 1$ and $last = -Max$. With $min\_rep = 3$, The valid segment $(D, p = 3, rep = 6, start = 3)$ is returned.

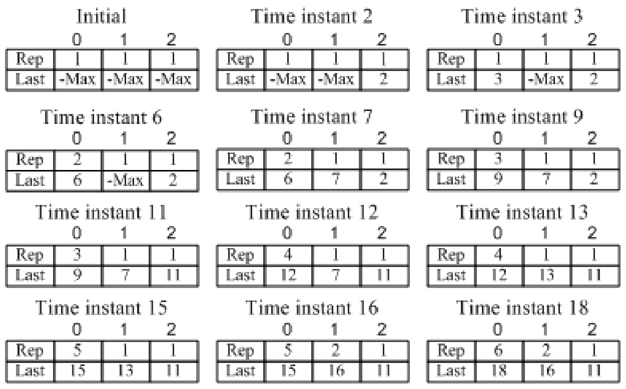
We analyze the time complexity and space complexity of the SPMiner below. The overall time for processing SPMiner for a given event e is $2 * n_e$ (PCD + HBV), where n_e is the number of occurrences of event e. For a given period length l, the time to find the singular periodic pattern for all events is hence $\sum_{\forall e} 2*n_e$ which is equivalent to two database scans. Let D denotes 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 structured 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. In this section, we will show how DFS enumeration can be used to discover valid segments for multi-event 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 multi-event pattern

With all valid segments discovered for single event 1-patterns, we can compose multi-event singular patterns by MPMiner as follows. Considers segments of the same period. Recall that a valid segment, S, is a 4-tuple $(EvtSet, p, rep, start)$ describing the event set, period,


Initial

	0	1	2
Rep	1	1	1
Last	-Max	-Max	-Max

Time instant 2

	0	1	2
Rep	1	1	1
Last	-Max	-Max	2

Time instant 3

	0	1	2
Rep	1	1	1
Last	3	-Max	2

Time instant 6

	0	1	2
Rep	2	1	1
Last	6	-Max	2

Time instant 7

	0	1	2
Rep	2	1	1
Last	6	7	2

Time instant 9

	0	1	2
Rep	3	1	1
Last	9	7	2

Time instant 11

	0	1	2
Rep	3	1	1
Last	9	7	11

Time instant 12

	0	1	2
Rep	4	1	1
Last	12	7	11

Time instant 13

	0	1	2
Rep	4	1	1
Last	12	13	11

Time instant 15

	0	1	2
Rep	5	1	1
Last	15	13	11

Time instant 16

	0	1	2
Rep	5	2	1
Last	15	16	11

Time instant 18

	0	1	2
Rep	6	2	1
Last	18	16	11

Figure 4. Execution process for event D with period 3

number of repetitions, and the start position of the segment. A segment discovered by SPMiner can also be considered as a 1-pattern of the form $(E_1, E_2, \ldots, E_p)$ where $E_i = EvtSet$ for $i = start\%p$, and $E_i = *$ otherwise. The index $start\%p$ is defined as the normalized offset. Two overlapped segments with the same offsets can form 2-event singular patterns if the repetition of the overlapping area is greater than min_rep. To discover i-event singular patterns, we can compose them from an $(i-1)$-event segment with 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 segments. Note that the end position of the segment can be determined by $start + (rep - 1) * p$. The same criteria work for combination of an $(i-1)$-event segment and a segment.

4.2.2 CPMiner for complex pattern

Discovering complex patterns from singular patterns has procedure similar to MPMiner. We refer to this procedure as CPMiner. CPMiner enumerates possible combinations of valid segments of the same period in depth-first order and check 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 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 discovered by MPMiner. Note that two segments with the same offset can only form a singular pattern and have been considered in MPMiner.

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 $(S_i.start - shift)\%p$, where $shift$ is the offset of the segment with the minimum end position.

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 less combinations because of the min_rep constraints for the overlapped 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 (anti-monotone property).

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

4.2.3 APMiner for asynchronous pattern

As noted in Definition 3.6, an asynchronous periodic pattern is defined by the existence of a valid sequence which is a set of non-overlapping 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 repetition is greater than $global\_rep$. For each enumeration, we try to extend current subsequence by examining one more segment. The ,if the start position of the 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 current sequence by max_dis, the remaining segments can be ignored since segments are ordered by their start position.

5 Experimental Results

For the purpose of performance evaluation, we use a 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 a normal distribution with average length P. Then L $(1 < L < P)$ positions are chosen for non-empty event sets. The average number of events for each singular pattern is set to $|I|$. The start position of a segment is randomly chosen from 1 to $D/4$. 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 $|D|$ time instants are used. 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 event in this time instant is less than T, the insufficient events are picked at random from the symbol set N. The default parameter is D50K-N1K-C5-L4-T10-I4-P20-Rep25-Dis50 ($min\_rep = 15$ and $L_{max} = 20$).

We only report performance comparison of SPMiner here due to space limitation. For comparison with SPMiner, we implement Yang et al.'s algorithm but omit phase C for it is designed to find the longest subsequence [6]. In order to handle multi-event temporal data, phase A and B are implemented using vertical format for each event. The general performance, the effect of parameters, and the scalability of our methods are considered here. The scalability of SPMiner is shown in Figure 5(a). The scaling with database size was linear and the running time for SPMiner is also better than LSI(A+B) (by a magnitude of 37 for $|D| = 150K$). The utility of PCD is demonstrated by the dashed line SPMiner(HBV) where HBV is executed without PCD. As we can see, even HBV itself has better performance than LSI by a magnitude of 15. The upper bound of SPMiner is close to SPMiner(HBV). Therefore, PCD is powerful pruning technique in single event pattern mining.

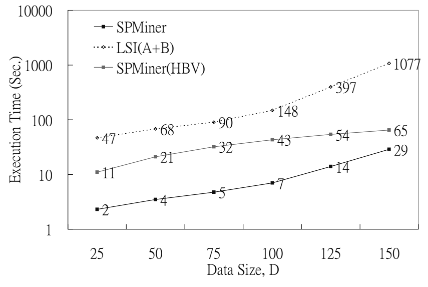
In Figure 5(b), the total running time for SPMiner(HBV) is linear to the average transaction size as analyzed in Section 4.1; while the running time for LSI increases dramatically since the distance-based pruning technique has comparatively less to prune. In summary, SPMiner is efficient and scalable in mining periodic pattern in multi-event temporal database.

6 Conclusion

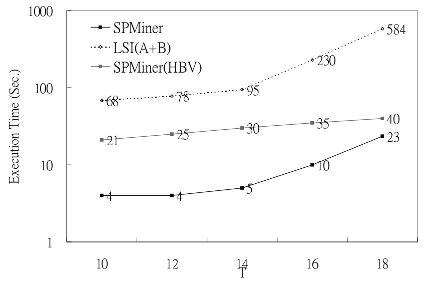
In this paper, a general model for asynchronous periodic pattern mining is defined. A four-phase algorithm which includes singular periodic pattern mining (SPMiner for 1-event, MPMiner for i-event), complex periodic pattern mining (CPMiner) and asynchronous sequence mining (APMiner) are devised to solve the problem. The experiments show that our algorithm is very efficient. Periodic pattern mining can be used not only for data characteristics summarization, it can also be applied for future periodicity predication. More research will be reported in the near future.

Acknowledgement

This work is sponsored by National Science Council, Taiwan under grant NSC92-2213-E-008-028.



(a)



(b)

Figure 5. Performance comparison.

References

[1] J. Han, G. Dong, and Y. Yin. Efficient mining paritial periodic patterns in time series database. In *Proc. of the 15th International Conference on Data Engineering,*, pages 106–115, 1999.

[2] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovering frequent episodes in sequences.

[3] B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proc. of the 14th International Conference on Data Engineering,*, pages 412–421, 1998.

[4] R. Srikant and R. Agrawal. Mining sequential patterns: Generalizations and performance improvements. In *Proc. of the 5th International Conference on Extending Database Technology,*, volume 1057 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 1996.

[5] A. K. H. Tung, J. Han H. Lu, and L. Feng. Breaking the barrier of transactions: Mining inter-transaction association rules,. In *Proc. of the International Conference on Knowledge Discovery and Data Mining,*, pages 297–301, 1999.

[6] J. Yang, W. Wang, and P.S. Yu. Mining asynchronous periodic patterns in time series data. In *Proc. of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 275–279, 2000.