Mining Periodic Patterns in Sequence Data

Kuo-Yu Huang and Chia-Hui Chang

Department of Computer Science and Information Engineering,
National Central University, Chung-Li, Taiwan 320
want@db.csie.ncu.edu.tw, chia@csie.ncu.edu.tw

Abstract. Periodic pattern mining is the problem that regards temporal regularity. There are many emerging applications in periodic pattern mining, including web usage recommendation, weather prediction, computer networks and biological data. In this paper, we propose a Progressive Timelist-Based Verification (PTV) method to the mining of periodic patterns from a sequence of event sets. The parameter *min_rep*, is employed to specify the minimum number of repetitions required for a valid segment of non-disrupted pattern occurrences. We also describe a partitioning approach to handle extra large/long data sequence. The experiments demonstrate good performance and scalability with large frequent patterns.

1 Introduction

Pattern mining plays an important role in data mining tasks, e.g. association mining, inter-transaction rule [6] sequential pattern and episode mining [4], etc. However, temporal regularity also plays an important roleIn association mining, we may specify a frequent pattern called the "Beer and Diaper" with support level of 10% and confidence level of 70%. If we consider the influence of time, we may find that the pattern, "Beer and Diaper", occurs at every Friday night for 20 weeks in a row. This cyclic association rule is a kind of periodic pattern which can be applied in period predictions. Besides, periodic patterns in biological data is also an important issues [3]. Periodic patterns in biological data may be a mechanism that provides regular arrays of spatial and function groups, useful for structural packing or for one to one interactions with target molecules. Periodic conservation of amino acids may be useful in structural packing of two or more polypeptide chains of the same or different proteins. Periodically placed amino acid side chains can also facilitate one to one interactions with target atoms showing similar periodicity" [3].

There have been a number of recent studies in periodic pattern mining. For example, cyclic association rules proposed by Ozden, et al. [5], partial periodic patterns by Han, et al. [1], and asynchronous periodic patterns by Yang, et al. [7, 2]. The so called **full periodicity** specifies the behavior of the time series at all points in period, while **partial periodicity** specifies the behavior at some but not all points in time. Ozden, et al. define the problem of discovering cyclic association rules that display regular cyclic variation over time [5]. 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. 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% 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 temporal data.

Since real life patterns are usually **imperfect**, Han et al. [1] 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 of 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 sequence. 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. [7] 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 periodicy. On the other hand, the disturbance between two valid segments has to be within some reasonable bound (max_dis). 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-pattern (called LSI). The second step then applies a level-wise search to generate the subsequences of i-patterns based on valid subsequences of all $(i-1)$-patterns with the same period length. Note that in order to discover the longest subsequence, a longer segment can be broken into small segments when two segments overlap. For some applications, such as biological data sequence, users might be more interested in the maximum segments that a periodic pattern can extend. In additions, the level-wise search for $(i-1)$-patterns might degrade for data sequences of event sets.

To address these problems, Huang and Chang in [2] propose a general model for mining asynchronous periodic patterns, where each valid segment is required to be of maximum and at least min_rep contiguous matches of the pattern. They decompose the problem into two subproblems, valid segment discovery and asynchronous subsequence composition. Valid segments include single-event 1-patterns, multi-event 1-patterns, and complex i-patterns ($i > 1$). Three algorithms SPMiner, MPMiner, and CPMiner are devised respectively. Finally, all valid segments with respect to a pattern can be combined to form an asyn-

chronous sequence by APMiner. Just like association rule mining, the computation load occurs at valid segment discovery. In this paper, we focus on the mining of valid segments and devise a progressive timelist-based verification algorithm (called PTV), which improve the performance of SPMiner and MPMiner. Experimental results show that this method offers better performance than the previous research when there are many periodic events.

The remaining parts of the paper are organized as follows. In Section 2, we define the problem of periodic pattern mining for sequence of event set. Section 3 presents our algorithm for mining periodic patterns from sequence data. Experiments and performances of the algorithm study are reported in Section 5. Complexity analysis and comparison to previous work are discussed 4. Our conclusion are presented in Section 6.

2 Problem Definition

In this section, we define the problem of periodic periodic mining. The problem definition is similar to [2]. Let E be a set of all events. An event set is a nonempty subset of E. A eventset sequence S is a set of time records where each time record is a tuple (tid, X) for time instant tid and event set X. A eventset sequence stored in form of (tid, X) is called horizontal format (see Fig. 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 $2 \leq j \leq l$.*

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.

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.*

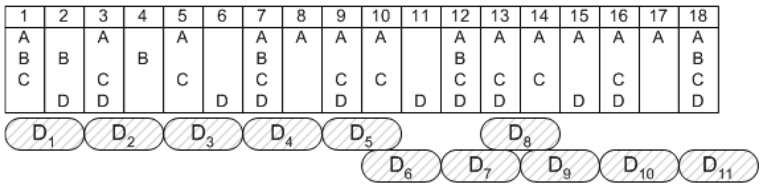

Fig. 1. The matches of (AC,*,*) in eventset sequence S.

In general, given a sequence of event sets and a pattern P, multiple matches of P may exist. In Fig. 1, $D_1, D_2, \ldots, D_{11}$ are 11 matches of $(AC, *, *)$.

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 if and only if it forms a contiguous subsequence of D. Here, k is referred to as the number of repetitions of this segment. A segment is* ***maximum*** *if there are no other contiguous matches at either end. 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.*

In Fig. 1, D_1, ..., D_5 are continuous and disjoint matches. Therefore, we can use $S_1 = \{(AC, *, *), 2, 5, 1\}$ to indicate a segment with period 2 starting from position 1 for 5 times. Note that D_1, D_2, D_3 and D_4 also form a segment but it is not maximum.

Definition 4. *A maximum segment S with respect to a periodic 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).*

For Fig. 1, if we set $min\_rep = 3$ and consider only pattern AC with period 2, there will be two valid segments S_1 (D_1, D_2, D_3, D_4 and D_5) and S_2 (D_6, D_7, D_9, D_{10} and D_{11}) returned. The problem is formulated as follows: given a eventset sequence and the parameters min_rep, the problem is to find all valid segment of periodic pattern with significant periods between L_{min} and L_{max} specified by the user.

3 Progressive Timelist-Based Verification

In this section, we explore a 2-phase algorithm, Progressive Timelist-Based Verification (PTV), for mining periodic patterns in eventset sequence. In the first phase, we modify the data structure of the SPMiner in [2] to discover single-event 1-patterns. In the second phase, we devise a timelist-based verification mechanism to combine all probable segments of multi-event 1-patterns.

The inputs to PTV include a vertical format database VD and the interesting period interval specified by L_{min} and L_{max}. The timelist in VD is maintained for each event. Essentially, PTV checks the time lists of each eventset for each possible period p ($L_{min} < p < L_{max}$) by a procedure called **PeriodicyCheck**. It starts by checking possible valid segments from the timelists for each single events (Phase I). If there exists a valid segment for an event, such events are enumerated in depth first order to form event sets. Duplicate enumerates are avoided by forcing an alphabetic order on the events. For each combined event set, the timelist is obtained by timelist intersection from the constituent events (Phase II). Again, the PeriodicyCheck procedure is applied to see if valid segments exist for the event set. Enumeration stops whenever no valid segment exists for an event set.

Given a timelist and period p, the task of **PeriodicyCheck** is to output valid segments with period p. This is implemented by keeping tracks of p independent segments in a data structure called $CSeg$, where each $CSeg$ records the start

Procedure of **PTV** (VD, L_{min},L_{max})
1. **/* Phase I */**
2. **for** p=L_{min} **to** L_{max} **do** $FList_p$= **NULL;**
3. **for each event** $E_i \in VD$ **do**
4. **if** ($|E_i.TimeList| < min\_rep$) **then continue;**
5. **for** p=L_{min} **to** L_{max} **do**
6. **if (PeriodicyCheck(**E_i**,** $E_i.TimeList$**,** p**))==true) then**
7. **Append** E_i **to** $FList_p$**;**
8. **/* Phase II */**
9. **for** p=L_{min} **to** L_{max} **do**
10. **if** ($|FList_p| > 1$) **then**
11. **for each event** $E_i \in FList_p$ **do**
12. $Node.Head = E_i$; $Node.Tail$ = all event $E_j \in FList_p$ $(j > i)$;
13. $Node.TimeList = E_i.TimeList$; DFS($Node$, p);

Procedure of **PeriodicyCheck (**$EvtSet$**,** $TimeList$**,** p**)**
1. Allocate data structure $Cseg[p]$;
2. **/* Initialization */**
3. **for** i=**1 to** p **do**
4. $CSeg[i].LP = -Max$; $CSeg[i].SP = -Max$;
5. **/* Validation */**
6. $Valid$ = false;
7. **for each time instant** $T_i \in TimeList$ **do**
8. $pos = T_i\%p$;
9. **if** ($T_i - CSeg[pos].LP) == p$) **then** $CSeg[pos].LP = T_i$**; continue;**
10. **if** ($CSeg[pos].LP - CSeg[pos].SP \geq (min\_rep - 1) * p$) **then**
11. **Output (**$EvtSet$**,** $(CSeg[pos].LP - CSeg[pos].SP)/p + 1$**,** $CSeg[pos].SP$**);**
12. $Valid$= true;
13. $CSeg[pos].SP = T_i$; $Cseg[pos].LP = T_i$;
14. **/* Rechecking */**
15. **for** $i = 1$ **to** p **do**
16. if ($CSeg[pos].LP - CSeg[pos].SP \geq (min\_rep - 1) * p$) **then**
17. **Output (**$EvtSet$**,** $(CSeg[pos].LP - CSeg[pos].SP)/p + 1$**,** $CSeg[pos].SP$**);**
18. $Valid = true$;
19. **return** $Valid$**;**

Procedure of **DFS**($Node$, p)
1. **for each** $E_i \in Node.Tail$
2. $newC.Head = Node.Head \cup E_i$;
3. $newC.Tail$ = Tail($Node.Tail$);
4. $newC.TimeList$ = Intersection($Node.TimeList$, $E_i.TimeList$);
5. **if (PeriodicyCheck(**$newC.Head$**,** $newC.TimeList$**,** p**)== true) then**
6. DFS($newC, p$);

Fig. 2. PTV: Progressive Timelist-Based Verification algorithm

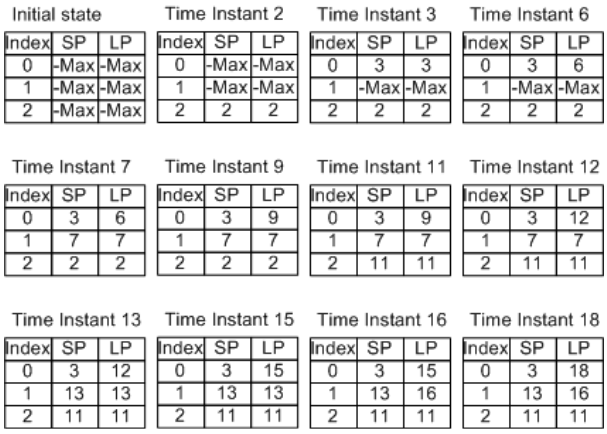

Fig. 3. Execution process for event D with period 3

position (SP) and last position (LP) for current segment. For each time instant T_i in the timelist, we compute the offset position $pos = T_i\%p$ and compare T_i to $CSeg[pos].LP$. If $T_i - CSseg[pos].LP$ is exactly p, it implies that this event set has occurred at $(T_i - p)$-th time instant. In this case, we update $Cseg[pos].LP$ by T_i. If otherwise, $T_i - CSseg[pos].LP$ is not p, it implies the last segment has been interrupted. In this case, output this segment if length of current segment ($CSeg[pos].LP - CSeg[pos].SP$) is greater than $(min\_rep - 1) * p$ and reset $CSeg[pos].SP$ and $CSeg[pos].LP$ to T_i (Step 6∼13 in the *PeriodicyCheck* in Fig. 2). Step 3∼4 of PeriodicyCheck initialize the allocated data structure. In order to check the unfinished segments, we recheck $CSeg$ again and output valid segments (Step 15∼18 in the *PeriodicyCheck*).

Let us use an example to illustrate this algorithm. Given the eventset sequence in Fig. 1, with parameters $min\_rep = 4$. The singular periodic pattern can be mined by the following steps.

- Phase I: For each period p, from L_{min} to L_{max}, use PeriodicyCheck procedure to check whether an event contains valid segments. Take period 3 and event D for example, three $CSeg$ are created and initialized with $SP = -Max$ and $LP = -Max$. For each time instant T_i in $D.TimeList$, the offset is computed by $pos = T_i\%3$. As shown in Fig. 3, time instant 3 has an offset at position 0, therefore $CSeg[0]$ is set with $SP = 3$ and $LP = 3$. Since time instant 6, 9, 12, 15 and 18 all have an offset 0, $CSeg[0].LP$ is updated five times. For time instant 2 and 11, both of them have an offset at position 2. However, time instant $11 - CSeg[2].LP$ does not equal 3, indicating an interrupt of the previous segment. Finally, segment $(D, 3, 6, 3)$ is output as valid. Since event D contains valid segment, it will be inserted into $FList_3$.
- Phase II: For each period p, from L_{min} to L_{max}, enumerate possible event sets from $FList_p$ in depth-first order by a recursive procedure **DFS**. The input to **DFS** is a $Node$ data structure which contains head event set, tail

event list, and the timelist for head event set. Take period 2 for example. $FList_2$ contains three events $\{A, C, D\}$. Assume an alphabetic order, $\{A, C\}$ is first enumerated by combining $Node.head = \{A\}$ with the first element from $Node.tail = \{C, D\}$. The timelist for $Node.head$ is the intersection of $A.TimeList$ and $C.TimeList$. With the timelist information, **PeriodicyCheck** procedure is then called to check if valid segments exist for this event set $\{A, C\}$. Since valid segments exist for this event set, **DFS** is called recursively with new node $(\{A, C\}, \{D\}, \{A, C\}.TimeList)$.

4 Discussion and Algorithm Comparison

In this section, we analysis the algorithms time/space complexity and discuss the solution when the sequence data is too long/large to fit in memory space.

4.1 Comparison

In this paper, PTV(I) is devised to discover all valid segments for each single event. Therefore, we compare the mining procedure of our proposed algorithm, PTV(I), with the LSI algorithm proposed in [7].

The overall time for processing PTV(I) for a given event e is n_e, 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 $\sum_{\forall e} n_e$ which is equivalent to one database scans. Let D denotes the number of time instants and T be the average number of events in each time instant. The database size can be represented by $D * T$. Consequently, the time complexity to discover all valid segments of 1-pattern for all periods is $O(S * T * L_{max})$ while $L_{min} = 1$. The data structured used for PTV(I) when processing an event is $CSeg$. The size of the data structure is a multiple of L_{max}, which can be reused for all events. Therefore, the space complexity is $O(L_{max})$.

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, LSI(A+B) is equivalent to PTV(I). We analyze the time complexity and space complexity of the PTV(I) below. The time complexity of LSI to discover the "longest" single event subsequence from a event sequence is $k * M * L_{max}$, where L_{max} is the maximum period length, M is the event sequence size and k is abbreviated for $min\_rep + max\_dis + L_{max}$ [7]. For a eventset sequence, the size of the database can be represented by $D * T$ for D time instants, 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 [7]. To discover valid segments as defined in this paper, the space of LSI can be approximated by $O(L_{max}^2 * N)$.

PeriodicyCheck in PTV is similar to the hash based validation (HBV) in SPMiner. In SPMiner, the access to $CSeg$ is two reads and write for Last and Rep updating; but the access to $CSeg$ needs only last position updating (Step 9 in the $PeriodicyCheck$). The MPMiner use a segment-based combination to generate

all multi-event 1-pattern. However, segment-based combination technology is not a robust method while the number of the valid segments are large. PTV(II) use a timelist-based validation method, but the time complexity is hard to analysis. Therefore, we compare PTV(II) and MPMiner by experiment result later.

4.2 Extra long/large sequence

Sometimes, the sequence data is too long/large to fit in memory space. In this case, we mine the periodic patterns by a partition-and-validation strategy. Firstly, the algorithm subdivides the extra-large sequence data into n non-overlapping partitions. Each partition can be handled in memory by PTV. Further, each partition is transformed into vertical format. For the fist partition, the valid segment can be mined by Initialization and Validation step in procedure PeriodicyCheck. For the succeeding partitions, the initial start (last) position is inherited from the last partition and valid segment pattern is explored by validation phase. In the final partition, valid segment is discovered by Validation and Rechecking step.

5 Experimental Results

To assess the performance of algorithm PTV, we conduct several experiments on a computer with a CPU clock rate of 1.13GHz and 256MB of main memory, the program is written by visual C++ in windows XP platform.

5.1 Biological data

We first apply PTV 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,*,*}, which reported in [3]. 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} occurring 11 times (approximately 46%) in the known periodic pattern. It may be a core pattern, since the partial slots of the pattern allow some mutations.

5.2 Synthetic data

For the purpose of performance evaluation, we use the same synthetic data as [2]. The default parameters of synthetic generator are data size D = 50K, event N = 1000, Avg. event in a time slot T = 10, potential pattern C = 3, pattern length L = 4, frequent event in a time slot I = 4, number of segment for each pattern S = 10. In order to make PTV more efficiency, we also use the PCD

pruning strategy to reduce unnecessary period checking [2]. We have run a series of experiments using PTV. The general performance, the effect of parameters, and the scalability of our methods are considered here. We start by looking at the performance of the PTV with parameter $min\_rep = 25$, $L_{min} = 1$ and $L_{max} = 20$.

5.3 Valid segments for single event 1-patterns

In this section, we compare the scalability of the three segment validation algorithms, including PTV(I), SPMiner and LSI. The scalability of PTV(I) is shown in Fig. 4(a). The total running time for PTV(I) increase smoothly, 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. The scalability for PTV(I) was also better than SPMiner. In Fig. 4(b) the total running time for PTV(I) is linear to T, whereas the running time for LSI increases dramatically. It is also evident that the number of pruned patterns is rapidly decreasing when the T increasing.

5.4 Valid segments for multi-event 1-patterns

Since LSI is devised for event sequence. Therefore, we demonstrate the efficiency of PTV(II) by comparing PTV(II) with MPMiner for multi-event 1-patterns. Fig. 4(c)(d) shows the execution time of these two methods. As we can see, MPMiner outperforms PTV(II) while S is small (S less than 15). However, when S is large, the execution time of PTV(II) increases smoothly. But the running time of MPMiner increases sharply. This is because the time complexity of MPMiner has an exponential relation to the number of segments in the worst case. In contrast, PTV(II) is comparably stable with respect to the number of segments. Fig. 4(d) shows the overall execution time of PTV(II) and MPMiner to find all valid segments for eventset. The x-axis shows the value of average event in the pattern, whereas the y-axis shows the overall execution time of PTV(II) and MPMiner. The execution time of PTV(II) increases more smoothly than MPMiner in Fig. 4(d).

6 Conclusion

In this paper, an efficient method for periodic pattern mining is defined. An algorithm progressive timelist-based validation (PTV) algorithm is devised to discover all valid segments in data sequence. The PTV algorithm the vertical database once and keeps only those timelists for events with valid segments. The experiments show that our algorithm outperform previous research. Periodic pattern mining can be used for data characteristics summarization and periodicity predication.

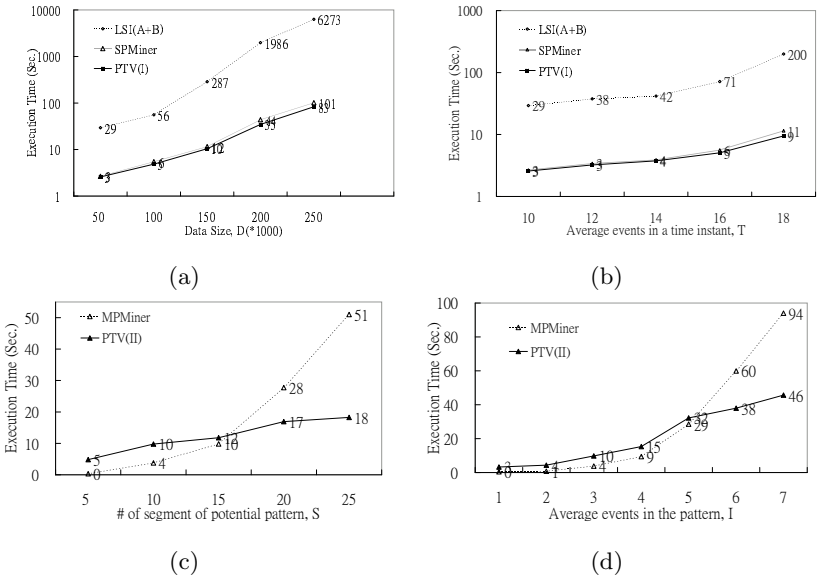

Fig. 4. Performance comparison

Acknowledgements

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

References

1. J. Han, G. Dong, and Y. Yin. Efficient mining paritial periodic patterns in time series database. In *Proceedings of the 15th International Conference on Data Engineering, (ICDE'99)*, pages 106–115, 1999.
2. K.Y. Huang and C.H. Chang. Asynchronous periodic patterns mining in temporal databases. In *Proceedings of the IASTED International Conference on Databases and Applications (DBA'04)*, pages 43–48, 2004.
3. 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*, 9:1203–1209, 2000.
4. H. Mannila, H. Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *Proceedings of the First International Conference on Knowledge Discovery and Data Mining (KDD'95)*, pages 210–215, 1995.
5. B. Ozden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proceedings of the 14th International Conference on Data Engineering, (ICDE'98)*, pages 412–421, 1998.
6. A. K. H. Tung, J. Han H. Lu, and L. Feng. Efficient mining of intertransaction association rules. *IEEE Transactions on Knowledge and Data Engineering*, 15(1):43–56, 2003.
7. J. Yang, W. Wang, and P.S. Yu. Mining asynchronous periodic patterns in time series data. *IEEE Transaction on Knowledge and Data Engineering*, 15(3):613–628, 2003.