Discovering Partial Periodic Patterns in Discrete Data Sequences

Huiping Cao, David W. Cheung, and Nikos Mamoulis

Department of Computer Science and Information Systems
University of Hong Kong
{hpcao, dcheung, nikos}@csis.hku.hk

Abstract. The problem of partial periodic pattern mining in a discrete data sequence is to find subsequences that appear periodically and frequently in the data sequence. Two essential subproblems are the efficient mining of frequent patterns and the automatic discovery of periods that correspond to these patterns. Previous methods for this problem in event sequence databases assume that the periods are given in advance or require additional database scans to compute periods that define candidate patterns. In this work, we propose a new structure, the abbreviated list table (ALT), and several efficient algorithms to compute the periods and the patterns, that require only a small number of passes. A performance study is presented to demonstrate the effectiveness and efficiency of our method.

1 Introduction

A *discrete data sequence* refers to a sequence of discrete values, e.g., events, symbols and so on. The problem of partial periodic pattern mining on a discrete data sequence is to find subsequences that appear periodically and frequently in the data sequence. E.g., in the symbol sequence "*abababab*", the subsequence "*ab*" is a periodic pattern. Since periodic patterns show trends in time series or event sequences, the problem of mining partial periodic patterns has been studied in the context of time series and event sequence databases ([1]-[3]).

Two essential sub-problems are the automatic discovery of periods and the efficient mining of frequent patterns. Given a period value, an Apriori-like algorithm is introduced in [3] to mine the frequent patterns. Han *et al.* in [2] propose a novel structure, max-subpattern tree, to facilitate counting of candidate patterns. This method outperforms the Apriori-like algorithm, but it assumes that the periods are given in advance, which limits its applicability. Berberidis *et al.* [1] has proposed a method that finds periods for which a data sequence may contain patterns. However, this method may miss some frequent periods and it requires a separate pass to scan the data sequence to compute the periods. What's more, for each symbol, it needs to compute, at a high cost, the circular autocorrelation value for different periods in order to determine whether the period is frequent or not.

We observe that a frequent pattern can be approximately expressed by an arithmetic series together with a support indicator about its frequency. In this paper, we propose a novel structure, the *abbreviated list table* (ALT), that maintains the occurrence counts of all distinct elements (symbols) in the sequence and facilitates the mining of periods and frequent patterns. Simultaneously, we present a fast $O(n)$ algorithm to identify periods from ALT.

The paper is organized as follows. Section 2 defines the mining problem formally. Section 3 presents the newly proposed approach. Section 4 includes performance evaluation of our methods. Finally, section 5 concludes the paper and proposes the future work.

2 Problem definition

Let **Domain** D be the set of elements that can be symbols, events, discretized locations, or any categorical object type. A discrete data sequence S is composed of elements from D and can be expressed as $S = e_0, e_1, ..., e_{n-1}$, where i denotes the relative order of an element and n is the length of S. Given a period T, a **periodic fragment** $s_i = e_{iT}, e_{iT+1}, ..., e_{(i+1)T-1}$, $(0 \leq i \leq \lfloor \frac{n}{T} \rfloor)$, is a subsequence of S, and $\lfloor \frac{n}{T} \rfloor$ is the number of fragments in S with respect to period T. Element $e_{iT+j}, (0 \leq j < T)$, in fragment s_i is at the j-th **period position**. There are T period positions, $0, 1, ..., T-1$, for a given period T.

Given a period T, a **T-period pattern** P is a sequence of elements p_0,p_1,...,p_{T-1}, $(0 \leq j < T)$, where p_j can be the wild card '$*$' or an element from D. If $p_j = *$, then any element from D can be matched at the j-th position of P. A periodic fragment $s_i = e_{iT}, e_{iT+1}, ..., e_{(i+1)T-1}$, $(0 \leq i \leq \lfloor \frac{n}{T} \rfloor)$, of a sequence S **matches** pattern $P = p_0, p_1, ..., p_{T-1}$ if $\forall j,\ 0 \leq j < T$, (1) $p_j = *$, or (2) $p_j = e_{iT+j}$.

A pattern is a **partial pattern** if it contains the element '$*$'. The **length** L of a pattern is the number of non-'$*$' elements in the pattern. We will call a length-L T-period pattern P an L-pattern if the period T is clear in the context. P' is a **subpattern** of a pattern P if it is generated from P by replacing some non-'$*$' elements in P by '$*$'. E.g.,'$a * c$' is a subpattern of the 3-pattern 'abc'. Similar to the period position of an element in a fragment, the **period position** of an element in pattern P is also the relative position of this element in the pattern. In the 3-period pattern '$a * *$', the period position of 'a' is 0.

The **support** of a T-period pattern P, denoted as $sup(P)$, in a sequence S is the number of periodic fragments that match P. A pattern P is **frequent** with respect to a support parameter min_sup, $(0 < min\_sup \leq 1)$, iff $sup(P) \geq \lfloor \frac{n}{T} \rfloor \times min\_sup$, (support threshold). If there exists a frequent T-period pattern, we say that T is a **frequent period**. Element e is a **frequent element** if it appears in a frequent pattern.

The **problem** of mining partial periodic pattern can now be defined as follows. Given a discrete data sequence S, a minimum support min_sup and a period window W, find:

(1) the set of frequent periods T such that $1 \leq T \leq W$; and

(2) all frequent T-period patterns w.r.t. min_sup for each T found in (1).

3 Mining using the Abbreviated List Table

This section describes the method for automatic discovery of frequent periods using the *Abbreviated List Table* (ALT), and a method that performs efficient mining of frequent patterns. In phase one, we scan the input sequence to construct an ALT that records the frequency of every element. In phase two, we use the max_subpattern tree [2] to mine the frequent patterns.

3.1 Abbreviated list table

If an element appears periodically for a period T, its positions in the sequence will form an arithmetic series, which can be captured by three parameters: period position, period and count(frequency). For example, in sequence S = '*bdcadabacdca*', the occurrences of 'a' for period = 2 are $\{3, 5, 7, 11\}$. We can represent them by $(1, 2, 4)$, where 1 is the period position of the occurrence of 'a' with period = 2, and 4 is the frequency of the corresponding pattern '$*a$'. For a given period T, for every element, we need to keep T representations: (0, T, $count_0$), ... , ($T-1$, T, $count_{T-1}$). We call these representations Abbreviated Lists (AL). The ALs of all the elements with respect to all periods bounded by a period window can be constructed at a single scan of the sequence.

The algorithm for maintaining the ALT is shown in Fig. 1a. ALT is a 3-dimensional table, the first dimension is the element's index in domain D, the second and third dimensions are period and period position, respectively.

```
Algorithm ALT1(ALT, W, S)
1. while(S still has elements){
2.    Read element e from S,
      //SeqPos is e's position in S;
3.    Get the index idx of e in D;
4.    for(p := 1; p ≤ W; p++){
5.       pos:=SeqPos mod p;
6.       ALT[idx][p − 1][pos]++;}}
7. //truncate sequence for all periods
```

Figure 1a. ALT Maintenance

```
Algorithm ALT2 (ALT, W, min_sup, n)
1. for (idx:=0; idx < |D|; idx++){
2.    get element e whose index equals to idx;
3.    for (p:=1; p ≤ W; p++){
4.       threshold:= ⌊n/p⌋ × min_sup;
5.       for (pos:=0;pos < p; pos++){
6.          if (ALT[idx][p − 1][pos] ≥ threshold)
7.             output p, pos and element e;
8. }}}
```

Figure 1b. Finding Periods and F_1

We now show an example for this algorithm. Let the data sequence S be "*abaaaccaae*" and period window W be 5. For the first 'a' at position 0, the counters at period position 0 of all the periods are incremented by 1. Upon seeing the second 'a' at position 2, for periods 1 and 2, the counters at period position 0 are incremented. For periods 3, 4, and 5, the counters at period position 2 are incremented. The process continues, until we process all the elements in S and have the values shown in Table 1.

While maintaining the Abbreviated List Table, we can compute the frequent periods and F_1 at any time moment against the length of the data sequence scanned. (F_1 is the set of size-1 frequent patterns). Fig. 1b shows the algorithm to compute the periods and F_1. We still take the ALT in Table 1 as example assuming $min\_sup = 0.8$. For $period = 2$, the threshold is $10/2 \times 0.8 = 4$, and 'a' at period position 0 is frequent because its count is 4. However, 'a' is not frequent at period position 1 because its count is only 2. So '$a*$' is a frequent pattern but '$*a$' is not, and 2 is a frequent period. Similarly, we can find other frequent periods 2, 4, 5 and their related F_1s.

Our method is more efficient than the circular autocorrelation method in [1] because of the following reasons: (1) We compute F_1 during the period discovery process in one pass. (2) Our method works well even when the length of data sequence n is unknown in advance. (3) We can find frequent periods directly from ALT.

3.2 Finding frequent patterns

For each frequent period found in step 1, the algorithm constructs the max-subpattern tree in step 2 using F_1 and the inverted lists. The lists of frequent elements are merged to reconstruct the fragments and they in turn are inserted into the max-subpattern tree. Finally, we get frequent patterns by traversing all the trees. We note two reasons for the superiority of the ALT-based algorithm over [2]. (1) This phase does not need to compute F_1. (2) Only the inverted lists for frequent elements are needed to build the tree. This requires less I/O compared with the original max-subpattern tree algorithm.

3.3 Analysis

Space complexity: Assume the period window is W, each element may have frequent periods from 1 to W. For each period, we need to record the occurrences at each period position. The number of counters in the ALT of an element is of $O(W^2)$. Given $|D|$ elements, the space required is $O(|D|W^2)$, which is independent of the sequence length. We expect that W and $|D|$ are in the order of hundreds in practice, thus the ALT can be accommodated in the main memory.

Time complexity: We need one scan on the sequence to build the ALT. Since the locations of ALT entries can be accessed in O(1) time, the time complexity to create the ALT is in the order of $O(n)$. The construction of the max-subpattern tree and the computation of the frequent patterns is also of $O(n)$.

4 Experiments

We compare our method with the algorithm proposed in [1]. We will use "ALT+tree" to denote our method, and use "Circular Autocorrelation+tree" to represent the method combining [1] and [2]. All the experiments were performed on a Pentium III 700MHz workstation with 4GB of memory, running Unix. A synthetic data generator is used to generate periodic object movements, with four parameters: period T, sequence length n, max pattern length l and probability p with which the object complies with the pattern.

Experimental comparison:

Efficiency: Given data sequence with parameters $n = 1$M, $T = 50$, $p = 0.8$ and $l = 25$, it's obvious to see that our method is much faster than [1] for fixed minimum support 0.6 from Fig. 2a. Moreover, the cost difference rises with the increase of the window size. Table 2 records the cost breakdown of the two methods for $W = 100$. Note that the cost difference is mainly attributed to the excessive cost of circular autocorrelation in discovering the periods. The finding of F_1 also contributes some cost difference as [1] needs one more scan on sequence to get F_1. The cost for building the max-subpattern tree in our method is less than that by scanning the whole sequence since we only need to access

the inverted lists for frequent elements. Fig. 2b shows that both methods have linear cost to the size of the database, due to the limited number of database scans, however, our method is much faster than the previous technique. For this experiment, we fix parameters $T = 100$, $p = 0.8$ and $l = 50$.

Table 3 lists the frequent periods found and the number of frequent patterns mined from the experiment of Fig. 2b. Since the parameter T used to generate data sequence is set to 100 for all sequences and the mining parameter for W is 100, only one frequent period 100 is found.

5 Conclusion

In this paper, we presented a new method to perform partial periodic pattern mining on discrete data sequences. Using the proposed ALT structure, we can find frequent periods and the set of 1-patterns during the first scan on data sequence. This step is much more efficient compared to a previous approach, which is based on circular autocorrelation. Further, frequent patterns are discovered by inserting fragments to the max-subpattern tree, using the inverted lists in order to avoid accessing irrelevant information. Our experiments show that the proposed technique significantly outperforms the previous approaches.

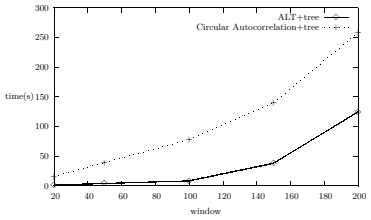

Figure 2a. Efficiency vs. W

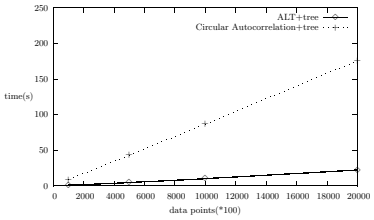

Figure 2b. Scalability

period pos	0	1	2	3	4
period=1	6				
period=2	4	2			
period=3	2	2	2		
period=4	2	0	1	2	
period=5	1	0	2	2	1

Table 1. ALT for a

time(s)	[1] +tree	ALT+tree
find period	71.34	5.59
find F_1	2.12	0
build trees	2.07	1.11
mine pat.	2.02	2.01

Table 2. Cost comparison

n	period	num of freq. pat.
100K	100	12
500K	100	42
1000K	100	76
2000K	100	133

Table 3.

References

1. C. Berberidis, I. P. Vlahavas, W. G. Aref, M. J. Atallah, and A. K. Elmagarmid. On the discovery of weak periodicities in large time series. In *Proc. 6th European Conf. on Principles and Practice of Knowledge Discovery in Databases*, 2002.
2. J. Han, G. Dong, and Y. Yin. Efficient mining of partial periodic patterns in time series database. In *Proc. of 15th Intl. Conf. on Data Engineering*, 1999.
3. J. Han, W. Gong, and Y. Yin. Mining segment-wise periodic patterns in time-related databases. In *Proc. of Intl. Conf. on Knowledge Discovery and Data Mining*, 1998.