Efficient Mining of Partial Periodic Patterns in Time Series Database

Jiawei Han [*]
School of Computing Science
Simon Fraser University
han@cs.sfu.ca

Guozhu Dong [†]
Department of Computer Science and Engineering
Wright State University
gdong@cs.wright.edu

Yiwen Yin
School of Computing Science
Simon Fraser University
yiweny@cs.sfu.ca

Abstract

Partial periodicity search, i.e., search for partial periodic patterns in time-series databases, is an interesting data mining problem. Previous studies on periodicity search mainly consider finding full periodic patterns*, where every point in time contributes (precisely or approximately) to the periodicity. However, partial periodicity is very common in practice since it is more likely that only some of the time episodes may exhibit periodic patterns.*

We present several algorithms for efficient mining of 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. The max-subpattern hit set property is a vital new property which allows us to derive the counts of all frequent patterns from a relatively small subset of patterns existing in the time series. We show that mining partial periodicity needs only two scans over the time series database, even for mining multiple periods. The performance study shows our proposed methods are very efficient in mining long periodic patterns.

Keywords. Periodicity search, partial periodicity, time-series analysis, data mining algorithms.

1. Introduction

Finding periodic patterns in time series databases is an important data mining task with many applications. Many methods have been developed for searching periodicity patterns in large data sets [8]. However, most previous methods on periodicity search are on mining *full periodic patterns*, where every point in time contributes (precisely or approximately) to the cyclic behavior of the time series. For example, all the days in the year *approximately* contribute to the season cycle of the year. A useful related type of periodic patterns, called *partial periodic patterns*, which specify the behavior of the time series at some but not all points in time, have not received enough attention. An example partial periodic pattern may state that Jim reads the Vancouver Sun newspaper from 7:00 to 7:30 every weekday morning but his activities at other times do not have much regularity. Thus, *partial periodicity* is a looser kind of periodicity than *full periodicity*, and it exists ubiquitously in the real world. The purpose of the current paper is to fill the gap by considering the efficient mining of partial periodic patterns.

Most methods for finding *full periodic patterns* are either inapplicable to or prohibitively expensive for the mining of *partial periodic patterns*, because of the mixture of periodic events and non-periodic events in the same period. For example, FFT (Fast Fourier Transformation) cannot be applied to mining partial periodicity because it treats the time-series as an inseparable flow of values. Some periodicity detection methods can detect some partial periodic patterns, but only if the period, and the length and timing of the segment in the partial patterns with specific behavior are explicitly specified. For the newspaper reading example, we need to explicitly specify details such as "find the regular activities of Jim during the half-hour after 7:00 for the period of 24 hours." A naive adaptation of such methods to our partial periodic pattern mining problem would be prohibitively expensive, requiring their application to a huge number of possible combinations of the three parameters of length, timing, and period.

Besides full periodicity search, there are many recent studies on time series data mining: Most concentrate on symbolic patterns, although some consider numerical curve patterns in time series. Agrawal and Srikant [3] developed an Apriori-like technique [2] for mining sequential patterns. Mannila et al. [10] consider frequent episodes in sequences, where episodes are essentially acyclic graphs

[*] Research was supported in part by research grants from the Natural Sciences and Engineering Research Council of Canada and the Networks of Centres of Excellence Program of Canada

[†] Part of this work was done while visiting Simon Fraser University during his sabbatical from University of Melbourne, Australia.

of events whose edges specify the temporal before-and-after relationalship but without timing-interval restrictions. Inter-transaction association rules proposed by Lu et al. [9] are implication rules whose two sides are totally-ordered episodes with timing-interval restrictions (on the events in the episodes and on the two sides). Bettini et al. [5] consider a generalization of inter-transaction association rules: these are essentially rules whose left-hand and right-hand sides are episodes with time-interval restrictions. However, unlike ours, periodicity is not considered in these studies.

Similar to our problem, the mining of cyclic association rules by Özden, et al. [12][1] also considers the mining of some patterns of a range of possible periods. Observe that cyclic association rules are partial periodic patterns with *perfect* periodicity in the sense that *each pattern reoccurs in every cycle*, with 100% confidence. The perfectness in periodicity leads to a key idea used in designing efficient cyclic association rule mining algorithms: As soon as it is known that an association rule R does not hold at a particular instant of time, we can infer that R cannot have periods which include this time instant. For example, if the maximum period of interest is ℓ_{max} and it is discovered that R does not hold in the first ℓ_{max} time instants, then R cannot have any periods. This idea leads to the useful "cycle-elimination" strategy explored in that paper. Since real life patterns are usually imperfect, our goal is not to mine perfect periodicity and thus "cycle-elimination" based optimization will not be considered here. [2]

An Apriori-like algorithm has been proposed for mining imperfect partial periodic patterns with a given (*single*) period in a recent study by two of the current authors [7]. It is an interesting algorithm for mining imperfect partial periodicity. However, with a detailed examination of the data characteristics of partial periodicity, we found that Apriori pruning in mining partial periodicity may not be as effective as in mining association rules.

Our study has revealed the following new characteristics of partial periodic patterns in time series: The Apriori-like property among partial periodic patterns still holds for any fixed period, but it does not hold for patterns between different periods. Furthermore, there is a strong correlation among frequencies of partial patterns.

The main contributions of this paper are as follows. We consider the efficient mining of partial periodic patterns, for a *single period* as well as for a *set of periods*. We propose several mining algorithms, 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. The max-subpattern hit set property is a vital new property which allows to derive the counts of all frequent patterns from a relatively small subset of patterns mined from the time series. We show that mining partial periodicity needs only two scans over the time series database, even for mining multiple periods. The performance study shows our proposed methods are very efficient. The proposed methods are also robust that can be applied in a variety of cases including mining multiple-level partial periodicity and mining partial periodicity with perturbation and evolution.

The remaining of the paper is organized as follows. In Section 2, concepts related to partial periodicity are introduced. In Section 3, methods for mining partial periodicity in regard to both single and multiple periods are studied. In Section 4, the implementation of a novel data structure, namely the max-subpattern tree, for facilitating the counting of the hit maximal patterns, and the derivation of the set of frequent patterns from the hit maximal patterns, are presented. In Section 5, a comparison of the performance of the proposed algorithms is reported. We conclude our study in Section 6.

2 Problem Definition

Assume that a sequence of n timestamped datasets have been collected in a database. For each time instant i, let D_i be a set of features derived from the dataset collected at the instant. Thus, the time series of features is repesented as,

$$S = D_1, D_2, \ldots, D_n.$$

Let L be the underlying set of features. We will also use the "don't care" character $*$, which can match any single set of features. We define a **pattern** $s = s_1 \cdots s_p$ as a non-empty sequence[3] over $(2^L - \{\emptyset\}) \cup \{*\}$. We will use $|s|$ to denote the length of s, and will say that $|s|$ is the **period** of the pattern s. Let the L-length of $s = s_1 \cdots s_p$ be the number of s_i which contains letters from L. A pattern with L-length i is also called an *i-pattern*. Moreover, a **subpattern** of a pattern $s = s_1 \cdots s_p$ is a pattern $s' = s'_1 \cdots s'_p$ such that s and s' have the same length, and $s'_i \subseteq s_i$ for every position i where $s'_i \neq *$. For example, the pattern $a * \{a,c\} de$ is of length 5 and it is of L-length 4 (i.e., it is a 4-pattern); and $a * \{a,c\} * *$ and $* * cde$ are two of the 2^5 subpatterns of $a * \{a,c\} de$.

The **frequency_count** and **confidence** of a pattern s in a time series $D_1, \ldots, D_n$ are defined as

$$frequency\_count(s) = |\{i \mid 0 \leq i < m,\ and$$

[1] It is important to point out that [12] concentrates on the elimination of candidate itemsets for the association rule mining algorithm, although the cycle-elimination strategy does lead to a small reduction on the number of patterns when we process the time series from left to right.

[2] Note that a modified strategy, where we stop considering certain patterns as soon as the length of the time series to be processed is not enough to make the confidence higher than the threshold, can be used.

[3] If s_i is a singleton we will omit the brackets, e.g., we write $\{a\}$ as a.

the string s *is true in* $D_{i|s|+1}\cdots D_{i|s|+|s|}\}|$,

and

$$conf(s) = \frac{frequency\_count(s)}{m},$$

where m is the maximum number of periods of length $|s|$ contained in the time series (i.e., m is the positive integer such that $m|s| \leq n < (m+1)|s|$). Each segment of the form $D_{i|s|+1}\cdots D_{i|s|+|s|}$, where $0 \leq i < m$, is called a **period segment**. We say a pattern $s = s_1 \cdots s_p$ is *true* in the period segment or the period segment *matches* s, if, for each position i, either s_i is $*$ or all the letters in s_i occur in the i^{th} set of features in the segment. Thus, if s' is a subpattern of s, then the set of sequences that can match s is a subset of sequences that can match s'.

Example 2.1 For example, $a*b$ is a pattern of period 3; its frequency count in the feature series $a\{b,c\}baebaced$ is 2; and its confidence is $\frac{2}{3}$, where 3 is the maximum number of periods of length 3. The frequency count of $a\{b,c\}*$ in $a\{b,c,d\}ea\{b,c\}aabc$ is also $\frac{2}{3}$. ▮

Similar to mining association rules [2], we say that a pattern is a **frequent partial periodic pattern** in a time series if its confidence is larger than or equal to a threshold, min_conf. The mining of frequent partial periodic patterns in a time series is to discover, possibly with some restrictions, all the frequent patterns of the series for one period or a range of specified periods. More specifically, the input to mining includes:

- A time series S.
- A specified period; or a range of periods specified by two integers low and $high$.
- An integer m indicating that the ratio of the lengths of S and the patterns must be at least m. This will ensure that the patterns mined would be of value to the application at hand.

Remark: Sometimes the derivation of the feature series from the original data series is quite involved, and the interaction of the periodic patterns with the derivation of features may lead to improved performance. Hence it is worthwhile to combine the mining of the features from the datasets with the mining of the patterns, as is the case for the mining of cyclic association rules [12]. For our work on the mining of frequent partial periodic patterns though, this interaction is not useful for achieving computational advantage and thus we will assume that we are dealing with the feature time series in our study.

3 Methods for mining partial periodicity in time series

In this section, we explore methods for mining partial periodicity in a time series, proceeding from mining partial periodicity for a *single* given period to mining partial periodicity for a specified range of periods (i.e., *multiple periods*).

3.1 Mining partial periodicity for single period

3.1.1 Single-period apriori method

A popular key idea used in the efficient mining of association rules is the *Apriori property* discovered in [2]: *If one subset of an itemset is not frequent, then the itemset itself cannot be frequent.* This allows us to use frequent itemsets of size i as filters for candidate itemsets of size $i+1$.

Interestingly, for each period p, the property supporting the Apriori "trick" still holds:

Property 3.1 [**Apriori on periodicity**] Each subpattern of a frequent pattern of period p is itself a frequent pattern of period p.

The proof is based on the fact that patterns are more restrictive than their subpatterns. Suppose s' is a subpattern of a frequent pattern s. Then s' is obtained from s by changing some set of letters to a subset or $*$. Hence s is more restrictive than s' and thus the frequency count of s' is greater than or equal to that of s. Thus s' is frequent as well.

An algorithm for mining partial periodic patterns for a given fixed period based on this Apriori "trick" was presented in [7]. We include a simplied version here for the sake of completeness.

Algorithm 3.1 [**Single-period Apriori**] Find all partial periodic patterns for a given period p satisfying a given confidence threshold *min_conf* in time-series S, based on the Apriori property 3.1.

Method.

1. Find F_1, the set of frequent 1-patterns of period p, by accumulating the frequency count for each 1-pattern in each *whole* period segment and selecting among them whose frequency count is no less than $min\_conf \times m$, where m is the maximum number of periods.
2. Find all frequent i-patterns of period p, for i from 2 up to p, based on the idea of Apriori, and terminate immediately when the candidate frequent i-pattern set is empty.

Analysis.

Number of scans over the time series. Step 1 of the algorithm needs to scan the time series S once. Step 2 needs to scan S up to $p-1$ times in the worst case. Thus the total number of scans is no more than the period p.

Space needed. (1) At Step 1, suppose there exist a total of f_i distinct features at positions $i, p{+}i, \ldots, (m{-}1)p{+}i$ in S, where m is the number such that $mp \leq |S| < (m{+}1)p$. We need $\Sigma_{i=1}^{p} f_i$ units of space [4] to hold the counts. In the worst case when every feature is distinct in the entire time series S, we need $\Sigma_{i=1}^{|S|} |D_i|$ units of space [5]. After Step 1, we only need $|F_1|$ units of space to keep F_1, the set of frequent 1-patterns in S. (2) At Step 2, the maximum number of candidate subpatterns that we may generate is $\left(\begin{array}{c}|F_1|\\ 2\end{array}\right) + \left(\begin{array}{c}|F_1|\\ 3\end{array}\right) + \cdots + \left(\begin{array}{c}|F_1|\\ |F_1|\end{array}\right) = 2^{|F_1|} - |F_1| - 1$. Considering that we still need $|F_1|$ space to keep the set of frequent 1-patterns, the total amount of space needed is $2^{|F_1|} - 1$ in the worse case in this computation. However, the average case should be much smaller than the worst case since if every feature is distinct in the time series, then there is no need to find periodic patterns. The existence of any periodicity in the time series will reduce the memory needed. ■

3.1.2 Single-period max-subpattern hit set method

Although the Apriori trick may reduce the search space in partial periodicity mining in a similar way as association rule mining, it is important to note that the data characteristics in the two cases are very different. In mining association rules, the number of frequent i-itemsets shrinks quickly as i increases because of the sparsity of frequent i-itemsets in a large transaction database. However, in mining partial periodicity, very often the number of frequent i-patterns shrinks slowly (when $i > 1$) as i increases. The slow speed of decrease in the number of frequent i-patterns is due to a strong correlation between frequencies of patterns and their subpatterns. We now illustrate this point.

Example 3.1 Suppose we have two frequent 1-patterns, $a*$ and $*b$, such that $conf(a*) = 0.9$ and $conf(*b) = 0.9$, in a time-series S. Then it must be the case that $0.8 \leq conf(ab) \leq 0.9$, as explained below. Since all period segments that match ab match both $a*$ and $*b$, $conf(ab) \leq 0.9$ holds. To derive the other inequality, let $\bar{a}$ denote the predicate that a letter is not a, similarly $\bar{b}$. The confidence of $\bar{a}*$ in S is at most 0.1, because $conf(\bar{a}*) = 1 - conf(a*)$. Similarly, $conf(*\bar{b}) \leq 0.1$. Since $conf(ab) \geq 1 - conf(\bar{a}*) - conf(*\bar{b})$, it follows that $conf(ab) \geq 0.8$. ■

The slow reduction of the set of candidate frequent i-patterns as i grows makes the Apriori pruning of Algorithm 3.1 less attractive. Is there a better way?

Obviously, the derivation of frequent 1-patterns is still an effective way to dramatically reduce the candidate set to be examined later because there are usually only a small number of features being frequent at a particular position but there could be a large number of features appearing in the position. This is especially true when the average number of features per position is larger than $\lceil \frac{1}{min\_conf} \rceil$. Thus our discussion will be focused on how to reduce the search effort after the set of frequent 1-patterns, F_1, is found.

Our key idea is based on the notions of max-patterns and hit patterns, defined next.

A **candidate (frequent) max-pattern**, C_{max}, is the maximal pattern which can be generated from F_1, the set of frequent 1-patterns. For example, if the frequent 1-pattern set is $\{a****, *b***, **c**, ***d*\}$, the candidate max-pattern is $abcd*$. Notice that a position in the candidate max-pattern may be allowed to have a disjunction of more than one non-$*$ letter. For example, if the frequent 1-pattern set is $\{a****, *b_1***, *b_2***, **c**, ***d*\}$, the candidate max-pattern is $a\{b_1, b_2\}cd*$.

Let the L-length of the candidate max-pattern, C_{max}, be $|C_{max}|$. A subpattern of C_{max} is **hit** in a period segment S_i of S if it is the maximal subpattern of C_{max} in S_i. For example, for $C_{max} = a\{b_1, b_2\}cd*$, the hit subpattern for a period segment $S_i = a\{b_1, b_2\}c_1\{d_1, d_2\}e$ is $a\{b_1, b_2\}***$, because it is true in S_i and none of its superpatterns $a\{b_1, b_2\}c**$, $a\{b_1, b_2\}*d*$, and $a\{b_1, b_2\}cd*$, is in S_i. The **hit set**, H, of a time series S is the set of all hit subpatterns of C_{max} in S.

The usefulness of hit max-patterns is: We can derive the complete set of partial periodic patterns, from the frequency counts of all the hit maximal subpatterns of C_{max}. This will be detailed below.

We would like to give an estimate of the buffer size needed in computation based on the idea of hit patterns. One upper bound of the buffer size is estimated in terms of m, the total number of periods in S. $|H|$, the size of the hit set in a time series S, should be no bigger than m, i.e., $|H| \leq m$. This is obvious since each period segment can generate at most one hit subpattern, and a hit subpattern may be hit in more than one period segment. The other upper bound of the buffer size is estimated in terms of the maximal number of patterns that can be generated from F_1, the set of frequent 1-patterns. Since each hit pattern of S is a subpattern of C_{max}, which is generated from F_1, similar to the analysis performed in Algorithm 3.1, the size of the set of subpatterns which can be generated from F_1 is $\left(\begin{array}{c}|F_1|\\ 1\end{array}\right) + \left(\begin{array}{c}|F_1|\\ 2\end{array}\right) + \cdots + \left(\begin{array}{c}|F_1|\\ |F_1|\end{array}\right) = 2^{|F_1|} - 1$. Therefore, $|H|$, the size of the hit set in a time series S, should be no bigger than $2^{|F_1|} - 1$. Combining both upper bounds, we have

[4] The unit of space is the space needed to hold the feature identifier and its associated count, and its size is usually 2-8 bytes, depending on the implementation.

[5] This is equal to the total space that the time series occupies.

Property 3.2 [**The bound of hit set**] The size of the hit set is bounded by the formula, $|H| \leq min\{m, 2^{|F_1|} - 1\}$, where m is the total number of periods in S, and F_1 is the set of frequent 1-patterns.

Using this formula, we can calculate the bound of the maximal buffer size needed in the processing: Given the set of frequent 1-patterns, F_1, the maximal (additional) buffer size needed for registering the counts of all the maximal subpatterns of C_{max} is $min\{m, 2^{|F_1|} - |F_1| - 1\}$.

This property is very useful in practice. For example, if we found 500 frequent 1-patterns when calculating *yearly* periodic patterns for 100 years, the buffer size needed is at most 100; on the other hand, if we found 8 frequent 1-patterns for calculating *weekly* periodic patterns for 100 years, the buffer size needed is at most $2^8 - 8 - 1 = 247$. We can always select the smaller one in estimating the maximal buffer size needed in computation.

Before turning to our hit-set based algorithm, we examine the probability distributions of maximal subpatterns of C_{max}.

Heuristic 3.1 [**Popularity of longer subpatterns**] The probability distribution of the maximal subpatterns of C_{max} is usually denser for longer subpatterns (i.e., with the L-length closer to $|C_{max}|$) than the shorter ones.

This heuristic can be observed in Example 3.1. From the example, we have $0.8 \leq prob\{maxsubpattern(ab) = ab\} \leq 0.9$, but $prob\{maxsubpattern(ab) = a*\} \leq 0.1$. In most cases, the existence of a short max-subpattern indicates that the nonexistence of some non-$*$-letter, which reduces the chance for the corresponding non-$*$ letter patterns to reach high confidence. Thus we have the heuristic.

This heuristics will imply that the number of nodes in the tree data structure of the next section is usually small. It is also useful for efficient buffer management: In order to reduce the overall cost of access, the longer subpatterns should be arranged to be more easily accessible (such as put in main memory) than the shorter ones.

We now present a main algorithm for mining partial periodic patterns for a given period, which is based on the discussions above.

Algorithm 3.2 [**Max-subpattern hit-set**] Find all the partial periodic patterns for a given period p in a time-series S, based on the max-subpattern hit-set, for a given *min_conf* threshold.

Method.

1. Scan S once to find F_1, the set of frequent 1-patterns of period p, using Step 1 of Algorithm 3.1. Form the candidate max-pattern, C_{max}, from F_1.
2. Scan S once. During the scan, for each period segment, if its hit set is nonempty, do the following: add the max-subpattern into the hit set buffer (with the associated count initialized to 1) if it is not already there; otherwise, increase the count of the max-subpattern by one. The hit set buffer is implemented in the form of a max-subpattern tree, a novel data structure, to be discussed in Section 4.
3. After the scan, derive the frequent patterns from the hit set. We will discuss how to implement the finding of the counts of the hit patterns and how to use these counts to derive the frequent patterns in Section 4. It turns out that both can be done efficiently.

Analysis.
Number of scans over the time series. The first step of the algorithm needs to scan S once. The second step needs to scan S one more time. Thus the total number of time-series scans is 2, independent of the period p.
Space needed. (1) The space needed for Step 1 is the same as Algorithm 3.1. After Step 1, we need $|F_1|$ units of space to keep F_1, the set of frequent 1-patterns in S. (2) At the second step, suppose there are $|F_1|$ frequent 1-patterns in S. According to Property 3.2, the total space needed for the hit set is at most $min\{m, 2^{|F_1|} - 1\}$, where m is the total number of periods in S. ■

In comparison with Algorithm 3.1, Algorithm 3.2 reduces the total number of scans of the time series from p (the length of the period) to 2, and it also uses much less buffer space in the computation in most cases. This can also be seen from the following observation: Suppose the hit subpattern for a period segment is $abcd$, which is not in the hit set yet. We need only one unit space to register the string and its count 1. However, for the Apriori technique, the candidate 2-patterns to be generated will be $\{ab**, a*c*, a**d, *bc*, *b*d, **cd\}$, 3-patterns to be generated will be $\{abc*, ab*d, a*cd, *bcd\}$, and the 4-patterns will be $\{abcd\}$, plus we have to update the count associated with each of them. Thus, it is expected that the max-subpattern hit set method may have better performance in most cases. We will compare the performance of the two algorithms in Section 5.

3.2 Mining partial periodicity with multiple periods

Mining partial periodicity for a *given* period covers a good set of applications since people often like to mine periodic patterns for natural periods, such as annually, quarterly, monthly, weekly, daily, or hourly. However, certain patterns may appear at some unexpected periods, such as every 11 years, or every 14 hours. It is interesting to provide facilities to mine periodicity for a range of periods.

To extend partial periodicity mining from one period to multiple periods, one might wish to extend the idea of Apriori to computing partial periodicity among different periods, that is, to use the patterns of small periods p as filters for candidate patterns of periods of the form kp for an integer $k > 1$. This will work if all frequent patterns of period kp are frequent patterns of period p. *Unfortunately, this is not the case.* For example, for the time series $abcdabceabcdabce$, $conf(**cd) = 1/2$, and $conf(cd) = 1/4$. Suppose the confidence threshold is 0.3. If we use from partial periodic patterns of period 2 as filter for candidate partial periodic patterns of period 4, we will miss the partial periodic pattern $**cd$.

Given that we cannot extend the Apriori "trick" to multiple periods, one obvious way to mine partial periodic patterns for a range of periods is to repeatedly apply the single-period algorithm for each period in the range.

Algorithm 3.3 [**Looping over single period computation**] Find all the partial periodic patterns for a set of periods in a given range of interest, $p_1, \ldots, p_k$, in the time-series S, with the given *min_conf* threshold.

Method.

1. for each period p_j in the range of interest (i.e., $p_1, \ldots, p_k$), apply Algorithm 3.2 ("max-subpattern hit-set") on period p_j.

Analysis.
Number of scans over the time series. Since each period will take 2 scans of the time series, the total number of scans of the time series is $2 \times k$.
Space needed. For computing partial periodicity for periods from p_1 to p_k, the space required is basically the sum of space for each p_j. Notice that the space required for initial Step 1 computation is still $\Sigma_{i=1}^{|S|} |D_i|$ in the worst case since the space once used in computation for period p_j, can be reinitialized and reused for computing other periods. But we need in total $\Sigma_{j=1}^{k} |F_1(p_j)|$ units of space to keep different sets of frequent 1-patterns, where $F_1(p_j)$ is the set of frequent 1-patterns in S derived for period p_j. Similarly, it takes at most $\Sigma_{j=1}^{k} min\{m_j, 2^{|F_1(p_j)|} - 1\}$ units of space to compute all, where m_j is the total number of periods p_j in S. ∎

Algorithm 3.3 provides an iterative method for mining partial periodicity for multiple periods. However, when the number of periods is large, we still need a good number of scans to mine periodicity for multiple periods. An improvement to the above method is to maximally explore the mining of periodicity for multiple periods in the same scan, which leads to the shared mining of periodicity for multiple periods, as illustrated below.

Algorithm 3.4 [**Shared mining of multiple periods**] Shared mining of all the partial periodic patterns for a set of periods in a given range of interest, $p_1, \ldots, p_k$, in time-series S, with the given *min_conf* threshold.

Method.

1. Scan S once, for all periods p_j in the range of interest, do the same as Step 1 in Algorithm 3.2.

   That is, for all periods p_j in the range of interest (i.e., $p_1, \ldots, p_k$), find $F_1(p_j)$, the set of frequent 1-patterns of period p_j, using the same Step 1 as in Algorithm 3.1. For each set of frequent 1-patterns of period p_j, form the candidate max-pattern, $C_{max}(p_j)$, from $F_1(p_j)$.

2. Scan S once, for all periods p_j in the range of interest, do the same as Step 2 in Algorithm 3.2.

   A similar process which will not be explained in detail.

Analysis.
Number of scans over the time series. The first step of the algorithm needs to scan S once. The second step needs to scan S one more time. Thus the total number of time-series scans is 2, independent of the period p.
Space needed. The total space required in the worst case is same as in Algorithm 3.3. ∎

Algorithm 3.4 explores shared processing at mining partial periodicity for multiple periods. The advantage of the method is that we only need two scans of time series for mining partial periodicity for multiple periods. The overhead of the method is that although it reduces the number of scans to 2, it will require more space in the processing of each scan than the multiple scan method because it needs to register the corresponding counts for each period p_j (for $1 < j < k$). However, since the shared features will share the space as well (with counts incremented), and there should be many shared features in periodicity search (otherwise, why mining periodicity?), the space required will hardly approach the worst case. Therefore, it should still be an efficient method in many cases for mining partial periodicity with multiple periods.

4 Derivation of all partial patterns

In this section, we examine the implementation considerations of our proposed algorithms. Algorithm 3.1 is an Apriori-like algorithm which can be implemented similarly as other Apriori-like algorithms for mining association rules (e.g. [2]). Algorithm 3.2 forms the basis for all the three remaining algorithms and requires new tricks to achieve efficiency, and thus our discussion is focused on its efficient implementation.

Algorithm 3.2 consists of two steps: Step 1, scan the time series once and find frequent 1-pattern set F_1; and

Step 2, scan the time series one more time, collect the set of the max-subpatterns hit in S, and derive the set of frequent patterns. The implementation of Step 1 is straightforward and has been discussed in the presentation of Algorithm 3.1. However, Step 2 is nontrivial and needs some good data structure to facilitate the storage of the set of max-subpatterns hit in S and the derivation of the set of frequent patterns.

A new data structure, called *max-subpattern tree*, is designed to facilitate the registration of the hit count of each max-subpattern and derivation of the set of frequent patterns, as illustrated in Figure 1. Its design is now outlined.

The max-subpattern tree takes the candidate max-pattern C_{max} as the root node, where each subpattern of C_{max} with one non-$*$ letter missing is a direct child node of the root. The tree expands recursively, according to the following rules. A node w, if containing more than 2 non-$*$ letters, *may* have a set of children, each of which is a subpattern of w with one more non-$*$ letter missing. Notice that a node containing only 2 non-$*$ letters will not have any children since every frequent-1 pattern is already in F_1. Importantly, we do not create a node if neither the node nor its descendant(s) containing more than 1 non-$*$ letter is hit in S [6]. Each node has a "count" field (which registers the number of hits of the current node), a parent link (which is nil for the root), and a set of child links; each child link points a child and is associated with a corresponding missing letter. A link can be nil when the corresponding child has not been hit.

Notice that a non-$*$ letter position of a max-subpattern in a max-subpattern tree may contain a set of letters, which matches the set of letters at the position in a period segment. For example, for $C_{max} = a\{b_1, b_2\} * d*$, the max-subpattern of the period segment $a\{b_1, b_2\}\{c_1, c_2\}d_3e_4$ is $a\{b_1, b_2\} * **$, and the segment will contribute one count to this node.

The update of the max-subpattern tree is performed as follows.

Algorithm 4.1 [**Insertion in the max-subpattern tree**] Insert a max-subpattern w found during the scan of S into the max-subpattern tree T.

Method.

1. Starting from the root of the tree, find the corresponding node by checking the missing non-$*$ letter in order.

   For example, for a max-pattern node $*b_1*d*$ in a tree with the root, $C_{max} = a\{b_1, b_2\} * d*$, there are two letters, a and b_2, missing. The node can be found by (1) following the $\bar{a}$ link (marked as "$\sim a$" in Figure 1) to $*\{b_1, b_2\} * d*$, and then (2) following the $\bar{b}_2$ link to $*b_1 * d*$, as shown in Figure 1.

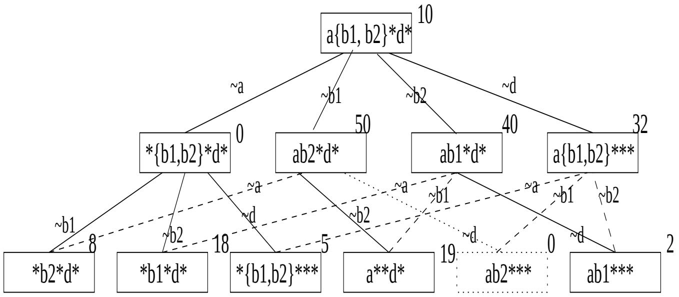

Figure 1. A max-subpattern tree to store the set of max-subpatterns hit in the time-series.

2. If the node w is found, increase its count by 1. Otherwise, create a new node w (with count 1) and its missing ancestor nodes (only those on the path to w, with count 0), if any, and insert it (or them) into the corresponding place(s) of the tree.

   For example, if the very first max-subpattern node found in S is $*b_1 * d*$ for $C_{max} = a\{b_1, b_2\} * d*$, we will create the node $*b_1 * d*$ (with count 1), after creating two ancestor nodes (with count 0): $w_1 = a\{b_1, b_2\} * d*$ (which is the root of the tree), and $w_2 = *\{b_1, b_2\} * d*$ (which is w_1's child, following the $\bar{a}$ link). The node $*b_1 * d*$ is w_2's child, following the $\bar{b}_2$ link.

Analysis.

Let the total number of non-$*$ letters in C_{max} be n_c. For a max-subpattern w containing n_w ($n_w > 1$) non-$*$ letters, we need to follow $n_c - n_w$ links to find the node and create at most $n_c - n_w + 1$ new nodes in the worst case. Therefore, the time complexity of node search and node creation will be less than n_c. Also, since each insertion of max-subpattern will create either only 0 node (when it hits) or less than n_c nodes, the total number of the nodes in the tree is less than $n_c \times |H|$, where $|H|$ is the size of the hit set. ■

In general, to insert a subpattern we need to both locate the position and update the count of the node if the node is found, or otherwise insert one or several new nodes.

Example 4.1 Let Figure 1 be the current max-subpattern tree T. To insert a (max)subpattern $ab_1 * **$ into the tree, we search the tree starting with the root, $C_{max} = a\{b_1, b_2\}*d*$. The first non-$*$ letter missing is b_2 and the second non-$*$ letter missing is d. Thus we first follow the $\bar{b}_2$ branch to node $ab_1 * d*$, and then follow the $\bar{d}$ branch. Since the node $ab_1 * **$ is located, its count is incremented by 1. ■

Before discussing the derivation of the set of frequent patterns, we need to introduce the concept of reachable ancestors. Since the traversal and creation of the children of a

[6] we show such a node $ab_2 * **$ using a dotted box in Figure 1.

node in the max-subpattern tree follow the non-$*$ letter position order, some of the ancestor nodes of a node may not be directly linked to a node. For example, in Figure 1, the node $a * *d*$ is linked to only one parent $ab_2 * d*$ but not the other $ab_1 * d*$ (note: this missing link is marked by a dashed line in the Figure).

In general, the **set of reachable ancestors** of a node w in a max-subpattern tree T is the set of all the nodes in T, which are proper superpatterns of w. It can be computed as follows: (1) derive a list w_m of missing letters from w based on C_{max}, which is roughly the position-wise difference, (2) the set of linked ancestors consists of those patterns whose missing letters form a proper prefix of w_m, and (3) the set of not-linked ancestors are those patterns whose missing letters form a proper sublist (but not prefix) of w_m.

Example 4.2 We compute the set of reachable ancestors for a node $* * *d*$ in a max-subpattern tree with root $C_{max} = a\{b_1, b_2\} * d*$. The list of missing non-$*$ letters is $[a, b_1, b_2]$. Thus, the set of linked ancestors is (1) $\bar{\emptyset}$ (missing nothing, which is the root); (2) $\bar{a}$ (i.e., missing a, which is the node $*\{b_1, b_2\} * de$); and (3) $\bar{a}\bar{b}_1$ (i.e., missing a, then missing b_1, which is the node $*b_2 * de$). The set of not-linked ancestors is: $*b_1 * d*$ (corresponding to the missing letter pattern $\bar{a}\bar{b}_2$), $ab_2 * d*$ (corresponding to $\bar{b}_1$), $a * *d*$ (corresponding to $\bar{b}_1\bar{b}_2$), and $ab_1 * d*$ (corresponding to $\bar{b}_2$). In other words, one can follow the links whose mark is not $\bar{d}$ in ordered way (to avoid visiting the same node more than once) and collect all the non-w nodes reached in T. ∎

Essentially there is a tree traversal for each fixed pattern, except that we do not visit a node and its descendants if the node is not an ancestor pattern of our current pattern.

The derivation of the frequent k-patterns is performed as follows.

Algorithm 4.2 [**Derivation of frequent patterns from max-subpattern tree**] The derivation of the frequent k-patterns for all k, given a max-subpattern tree T, by an Apriori-like technique.

Method.

1. The set of frequent 1-patterns F_1 is derived in the first scan of Algorithm 3.2.

2. The max-subpattern tree T is derived in the second scan of Algorithm 3.2. The set of frequent k-patterns ($k > 1$) is derived as follows.

   for $i := 2$ to $|F_1|$ do $\{$

   - derive candidate patterns with L-length i from frequent patterns with L-length $(i-1)$ by "$(i+1)$-way join".

   - scan tree T to find frequency counts of these candidate patterns and eliminate the non-frequent ones. Notice that the frequency count of a node is the sum of the count of itself and those of all of its reachable ancestors. If the derived frequent i-pattern set is empty, return.

   $\}$

Analysis.

Let the total number of non-$*$ letters in C_{max} be n_c. As shown in the analysis of Algorithm 4.1, the time complexity for searching a node is less than n_c. Since there are at most $2^{n_c} - n_c$ nodes to be generated from the max-pattern tree T (including all the missing descendants), and there are at most $|H|$ reachable ancestors in T, where $|H|$ is the size of the hit set, the worst case time complexity for derivation of all the frequent patterns is O($n_c \times 2^{n_c} \times |H|$), i.e., proportional to c_n and the size of the hit set, but exponential to n_c (i.e., proportional to the size of the tree that can be generated by C_{max}). Since an infrequent node will reduce the number of candidates to be generated in the future rounds, the real processing cost is usually much smaller than the cost in the worst case. ∎

We illustrate how to derive the frequent k-patterns for $k > 1$ from the max-subpattern tree T.

Example 4.3 Let Figure 1 be the derived max-subpattern tree T, and $min\_conf \times m = 45$. We can traverse the max-subpattern tree to find all the frequent k-patterns for $k > 1$ as follows. Starting at level 2, we have the following frequent patterns: $\{*b_2 * d*$ (68), $*b_1 * d*$ (68), $*\{b_1, b_2\} * **$ (47), $a * *d*$ (119), $ab_2 * **$ (92), $ab_1 * **$ (84)$\}$. We show the derivation of $*b_2 * d*$ (68) here: since the list of missing letters in this node is $[ab_1]$, its set of reachable ancestors is $\{\bar{\emptyset}$, $\bar{a}$, $\bar{b}_1\}$, and thus its frequent count = 10 + 0 + 50 + 8 (itself) = 68. Since level-2 has no infrequent nodes, we search all the nodes at level-1 and have the following frequent patterns: $\{ab_2 * d*$ (60), $ab_1 * d*$ (50)$\}$; Since there is one node infrequent, level-0 (root) has no frequent patterns. Notice although we only saved one node computation in this case, it will save much more when the tree is large and there are more missing nodes. ∎

From the above example, one can see that there are many frequent k-patterns with small k that can be generated from a max-subpattern tree. In practical applications, people may only be interested in the **set of maximal frequent patterns** instead of all frequent patterns, where a set of maximal frequent patterns is a subset of the frequent pattern set and every other pattern in the set is a subpattern of an element in the set. For example, if the set of frequent k pattern (for $k > 1$) is $\{ab * *, *bc*, a * c*, abc*\}$, the set of maximal frequent patterns is $\{abc*\}$.

If a user is interested in deriving the set of maximal frequent patterns, the MaxMiner algorithm developed by Bayardo [4] is a good candidate. The success of this algorithm stems from generating new candidates by joining frequent itemsets and looking head. However, it still requires to scan S up to period p times in the worst case. The mixture of max-subpattern hit set method and the MaxMiner can get rid of this problem and will be more efficient than pure MaxMiner. The details of the new method will be examined in future research.

5 Performance study

In this section we report a performance study which compares the performance of the periodicity mining algorithms proposed in this paper. In particular, we give a performance comparison between the single-period Apriori algorithm (Algorithm 3.1) (or simply called Apriori), and the max-subpattern hit-set algorithm (Algorithm 3.2) (or simply hit-set) applied to a single period.

This comparison indicates that there is a significant gain in efficiency by max-subpattern hit-set over Apriori. Since there is more gain when applied to multiple periods by using max-subpattern hit-set, it is clear that max-subpattern hit-set is the winner.

The performance study is conducted on a Pentium 166 machine with 64 megabytes main memory, running in Windows/NT. The program is written in Microsoft/VisualC++.

5.1 Testing Databases

Each test time series is a synthetic time-series databases generated using a randomized periodicity data generation algorithm. From a set of features, potentially frequent 1-patterns are composed. The size of the potentially frequent 1-patterns is determined based on a Poisson distribution. These patterns are generated and put into the time-series according to an exponential distribution.

LENGTH$_S$	the length of time series
p	a period
MAX-PAT-LENGTH	the maximal L-length of frequent patterns
$\lvert F_1\rvert$	the number of frequent 1-patterns

Table 1. Parameters of synthetic time series

The basic parameters used to generate the synthetic databases are listed in Table 1. The parameters of LENGTH$_S$ (the length of time series) and p (a period) are independently chosen. The parameters of MAX-PAT-LENGTH (the maximal L-length of frequent patterns) and $|F_1|$ (the number of frequent 1-patterns) are for a fixed p, and they are controlled by the choice of some appropriate confidence threshold. We found that other parameters, such as the number of features occurring at a fixed position and the number of features in the time series, do not have much impact on the performance result and thus they are not considered in the tests.

5.2 Performance comparison of the algorithms

Figure 2 shows there is a significant efficiency gain by max-subpattern hit-set over Apriori. In this figure, the maximal pattern length (the maximal L-length of frequent partial periodic patterns) grows from 2 to 10. The other parameters are kept constant: $p = 50$ and $|F_1| = 12$. We run two sets of tests, one with the length of the time series being $100,000$ and the other being $500,000$. As we can see, the running time of max-subpattern hit-set is almost constant for both cases, while Apriori is almost linear. When MAX-PAT-LENGTH is ≥ 8, the gain by max-subpattern hit-set over Apriori is about double. We expect this gain will increase for larger MAX-PAT-LENGTH.

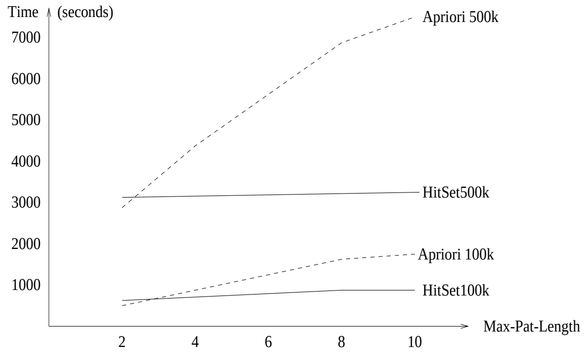

Figure 2. Performance gain when MAX-PAT-LENGTH increases: $p = 50, |F_1| = 12$.

It is important to note that, the gain shown in Figure 2 is done by keeping everything in memory, and by considering only one period. In general, this will be unlikely the case, and max-subpattern hit-set will perform even better than Apriori for the following reasons:

- In general, the time series of features may need to be stored on disk, due to factors such as each D_i may contain thousands of features and the length of the time series can be longer. When the time series is stored on disk, there would be a large amount of extra disk-IO associated with Apriori, but not with max-subpattern hit-set since it only requires two scans. Even when the time series is not stored on disk, Apriori will need to go over this huge sequence many more times than max-subpattern hit-set. Thus max-subpattern hit-set will be far better than Apriori.

- When there are a range of periods to consider, max-subpattern hit-set can find all frequent patterns in two scans but Apriori will require many more scans, depending on the number of periods and the L-length of the maximal frequent patterns. Hence max-subpattern hit-set will be again far better than Apriori.

6 Conclusions

We have studied efficient methods for mining partial periodicity in time series database. *Partial periodicity*, which associates periodic behavior with only a *subset* of all the time points, is less restrictive than *full periodicity* and thus covers a broad class of applications.

By exploring several interesting properties related to partial periodicity, including the Apriori property, the max-subpattern hit set property, and shared mining of multiple periods, a set of partial periodicity mining algorithms are proposed, with their relative performance compared. Our study shows that the max-subpattern hit set method, which needs only two scans of the time series database, even for mining multiple periods, offers excellent performance.

Our study has been confined to mining partial periodic patterns in one time series for categorical data with single level of abstraction. However the method developed here can be extended for mining multiple-level, multiple-dimensional partial periodicity and for mining partial periodicity with perturbation and evolution.

For mining numerical data, such as stock or power consumption fluctuation, one can examine the distribution of numerical values in the time-series data and discretize them into single- or multiple- level categorical data. For mining multiple-level partial periodicity, one can explore level-shared mining by first mining the periodicity at a high level, and then progressively drilling-down with the discovered periodic patterns to see whether they are still periodic at a lower level.

Perturbation may happen from period to period which may make it difficult to discover partial periodicity in many applications. For mining partial periodicity with perturbation, one method is to slightly enlarge the time slot to be examined. Partial periodic patterns with minor perturbation are likely to be caught in the generalized time slot. Another method is to include the features happening in the time slots surrounding the one being analyzed. We can further employ regression technique to reduce the noise of perturbation.

There are still many issues regarding partial periodicity mining which deserve further study, such as further exploration of shared mining for mining periodicity with multiple periods, mining periodic association rules based on partial periodicity, and query- and constraint- based mining of partial periodicity [11]. We are studying these problems and implementing our algorithms for mining partial periodicity in a data mining system and will report our progress in the future.

References

[1] R. Agrawal, G. Psaila, E. L. Wimmers, and M. Zait. Querying shapes of histories. In *Proc. 21st Int. Conf. Very Large Data Bases*, pages 502–514, Zurich, Switzerland, Sept. 1995.

[2] R. Agrawal and R. Srikant. Fast algorithms for mining association rules. In *Proc. 1994 Int. Conf. Very Large Data Bases*, pages 487–499, Santiago, Chile, September 1994.

[3] R. Agrawal and R. Srikant. Mining sequential patterns. In *Proc. 1995 Int. Conf. Data Engineering*, pages 3–14, Taipei, Taiwan, March 1995.

[4] R. J. Bayardo. Efficiently mining long patterns from databases. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, pages 85–93, Seattle, Washington, June 1998.

[5] C. Bettini, X. Sean Wang, and S. Jajodia. Mining temporal relationships with multiple granularities in time sequences. *Data Engineering Bulletin*, 21:32–38, 1998.

[6] J. Han and Y. Fu. Discovery of multiple-level association rules from large databases. In *Proc. 1995 Int. Conf. Very Large Data Bases*, pages 420–431, Zurich, Switzerland, Sept. 1995.

[7] J. Han, W. Gong, and Y. Yin. Mining segment-wise periodic patterns in time-related databases. In *Proc. 1998 Int'l Conf. on Knowledge Discovery and Data Mining (KDD'98)*, New York City, NY, August 1998.

[8] H. J. Loether and D. G. McTavish. *Descriptive and Inferential Statistics: An Introduction*. Allyn and Bacon, 1993.

[9] H. Lu, J. Han, and L. Feng. Stock movement and n-dimensional inter-transaction association rules. In *Proc. 1998 SIGMOD Workshop on Research Issues on Data Mining and Knowledge Discovery (DMKD'98)*, pages 12:1–12:7, Seattle, Washington, June 1998.

[10] H. Mannila, H Toivonen, and A. I. Verkamo. Discovering frequent episodes in sequences. In *Proc. 1st Int. Conf. Knowledge Discovery and Data Mining*, pages 210–215, Montreal, Canada, Aug. 1995.

[11] R. Ng, L. V. S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proc. 1998 ACM-SIGMOD Int. Conf. Management of Data*, pages 13–24, Seattle, Washington, June 1998.

[12] B. Özden, S. Ramaswamy, and A. Silberschatz. Cyclic association rules. In *Proc. 1998 Int. Conf. Data Engineering (ICDE'98)*, pages 412–421, Orlando, FL, Feb. 1998.