The Journal of Systems and Software 84 (2011) 1638–1651

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

# New and efficient knowledge discovery of partial periodic patterns with multiple minimum supports

Shih-Sheng Chen [a], Tony Cheng-Kui Huang [b,*], Zhe-Min Lin [c]

[a] Department of Information Management, National Chin-Yi University of Technology, No.57, Sec. 2, Zhongshan Rd., Taiping Dist., Taichung 411, Taiwan, ROC
[b] Department of Business Administration, National Chung Cheng University, 168 University Road, Min-Hsiung, Chia-Yi, Taiwan, ROC
[c] Department of Information Management, Tatung University, No.40, Sec. 3, Zhongshan N. Rd., Taipei City 104, Taiwan, ROC

ARTICLE INFO

*Article history:*
Received 20 June 2009
Received in revised form 11 April 2011
Accepted 12 April 2011
Available online 22 April 2011

*Keywords:*
Data mining
Partial periodicity
FP-tree
Multiple minimum supports

ABSTRACT

The problem of mining partial periodic patterns is an important issue with many applications. Previous studies to find these patterns encounter efficiency and effectiveness problem. The efficiency problem is that most previous methods were proposed to find frequent partial periodic patterns by extending the well-known Apriori-like algorithm. However, these methods generate many candidate partial periodic patterns to calculate the patterns' supports, spending much time for discovering patterns. The effective problem is that only one minimum support threshold is set to find frequent partial periodic patterns but the results is not practical for real-world. In real-life circumstances, some rare or specific events may occur with lower frequencies but their occurrences may offer some vital information to be referred in decision making. If the minimum support is set too high, the associations between events along with higher and lower frequencies cannot be evaluated so that significant knowledge will be ignored. In this study, an algorithm to overcome these two problems has been proposed to generating redundant candidate patterns and setting only one minimum support threshold. The algorithm greatly improves the efficiency and effectiveness. First, it eliminates the need to generate numerous candidate partial periodic patterns thus reducing database scanning. Second, the minimum support threshold of each event can be specified based in its real-life occurring frequency.



## 1. Introduction

The problem of mining periodic patterns is one of the important issues in data mining. Its purpose is to discover regularity in time series data or sequence data. Finding periodic patterns is a significant task with many business applications, such as tracing the regularities of companies' stock prices rising every week, reordering points in inventory management happening in every month, and analyzing the sales volumes of sales promotions made every weekend. Han et al. (1998) divided periodic patterns into two types of patterns, full periodic and partial periodic patterns. The former considers that every point in the period contributes to the cycle behavior of the time series, such as all the hours (days) in a day (year). According to the latter, some but not all points in the period contribute to the cycle behavior of the time series. Yang et al. (2001), moreover, divided the task of mining periodic patterns into support and information models. The former is concerned with whether a pattern is frequent, whereas the latter explores whether a pattern is expected to occur frequently based on some prior knowledge or by chance. Since using the support model to mine partial periodic patterns is more popular in business applications, the supportive move will be made in the following section.

Han et al. (1998) were the first to apply an Apriori-like algorithm to find partial periodic patterns. To efficiently mine partial periodic patterns, Han et al. (1999) explored the max-subpattern hit set property. However, the max-subpattern hit set algorithm proposed by Han et al. (1999) may lead to two issues, efficiency and effectiveness problems, which will be discussed below.

Originally, the Apriori-like algorithm (Agrawal and Srikant, 1994) adopted a candidate generation-and-test strategy. First, it finds the frequent itemsets by scanning the database, and calculates the occurrences of itemsets. If the occurrence of an itemset exceeds a given threshold, called minimum support, the itemset can be defined frequent (i.e. large). Afterwards, all frequent itemsets are treated as seeds to generate the candidate itemsets in the next pass. The other candidate itemsets are generated and tested iteratively to find the complete frequent itemsets. If there are no frequent itemsets to generate candidate itemsets in the next pass, the algorithm is terminated. In summary, this type of the algorithm leads problems of efficiency because the Apriori-like approach generates numerous candidate patterns and the procedure is time-consuming. Fortunately, Han et al. (2000) proposed a new data structure, FP-tree, and

* Corresponding author. Tel.: +886 5 2720411x34319; fax: +886 5 2720564.
*E-mail address:* bmahck@ccu.edu.tw (T.C.-K. Huang).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.04.022

an algorithm, FP-growth, by mining frequent itemsets to avoid generating any candidate itemsets. The performance of their algorithm is more efficient than that of the Apriori-like algorithm.

Traditionally, the problem of mining partial periodic patterns only considers setting one minimum support threshold (min_sup) for all events. Using the single min_sup implies that all events occur with similar frequencies in the database. However, this approach is not workable in practical applications because there may be problems with effectiveness. The set of example events representing the stock statuses includes rising dramatically, rising marginally, remaining unchanged, falling marginally or falling dramatically. In general, the stock price rises marginally, remains unchanged or falls marginally very frequently; however it seldom rises dramatically or falls dramatically. For instance, a company has different daily stock prices each week. The stock price of the company is defined an "event" observed and a week is defined as a "period". Event a is set as rising marginally and event b is falling dramatically. Additionally, event a occurs four times on Mondays and event b occurs twice on Fridays during the 5-week observations period. Traditionally, the minimum support is set at 80% and only a partial periodic pattern, event a, is found. Event b will be ignored, because its occurrence fails to exceed the 80% threshold. However, the relationship between events a and b within this period will not be found. Therefore, valuable information will not be provided for making worthy business investments. If a lower min_sup is set for event b, the problem of detecting event b can be remedied. But, many meaningless frequent patterns may be found and causes the combination explosion problem. To solve the rare item problem, Liu et al. (1999) proposed a model allowing users to specify multiple minimum supports to reflect different natures and frequencies of items (events in our study). Hence, this idea is applied for mining partial periodic patterns.

As discussed above, MSApriori algorithm proposed by Liu et al. (1999), applying the concept of the Apriori-like algorithm, also suffers from inferior performance. To overcome the efficiency and effectiveness problems in mining partial periodic patterns, a novel and efficient algorithm, the PFP-growth algorithm, has been proposed to find partial periodic patterns with multiple minimum supports. This method originates from the FP-tree's concept (Han et al., 2000) such that the advantages of both FP-growth and MSApriori algorithms can be retained. The proposed approach offers two advantages: (1) it is not necessary to scan databases many times to generate enormous candidate partial periodic patterns and (2) the minimum support threshold of each event can be specified by a user depending on its real-life occurring frequency.

This paper is organized as follows. In Section 2, we review three algorithms related to our study. Section 3 presents the PFP-tree structure and its construction method. Section 4 develops the approach called the PFP-growth algorithm. Section 5 shows our performance evaluation. Finally, conclusions are drawn in Section 6.

## 2. Related works

Since the introduction of the mining of partial periodic patterns with multiple minimum supports stems from previous research (Liu et al., 1999; Han et al., 1999), three algorithms will be briefly reviewed in this section. We introduce the MSAprori algorithm in Section 2.1, the FP-growth algorithm in Section 2.2, and the max-subpattern hit set algorithm in Section 2.3.

### 2.1. The MSApriori algorithm

The MSApriori algorithm was proposed by Liu et al. (1999). This model extends the existing association rule model to allow users to specify multiple minimum supports to reflect different natures and frequencies of items. Moreover, it enables users to find rare item rules without producing a huge number of meaningless rules. In this model, the definition of minimum support is changed, and each item in the database has its minimum support threshold, which is expressed in terms of *minimum item supports* (MIS). By providing different MIS values for different items, users can effectively express different support requirements for different rules. For example, consider the following items in a database, *bread*, *shoes*, and *clothes*. The MIS values are specified as follows: MIS(*bread*) = 2%, MIS(*shoes*) = 0.1%, MIS(*clothes*) = 0.2%. If the support of itemset {*clothes*, *bread*} is 0.15%, the itemset {*clothes*, *bread*} is infrequent because the MIS value of itemset {*clothes*, *bread*} is equal to min[MIS(*clothes*), MIS(*bread*)] = 0.2%, which is larger than 0.15%. On the other hand, if the support of itemset {*shoes*, *bread*} is 0.15%, the itemset {*shoes*, *bread*} is frequent because its MIS value is equal to min[MIS(*shoes*), MIS(*bread*)] = 0.1%, which is not larger than 0.15%. Therefore, setting different MIS for different items will result in different thresholds to find frequent patterns.

Mining association rules typically consist of two steps: (1) finding all frequent itemsets and (2) generating association rules using the frequent itemsets. When there is only one minimum support, the above two steps satisfy the *downward closure property*. That is, if a set of items can satisfy the minimum support, all its subsets will also satisfy the minimum support. For example, consider four items, a, b, c, and d, in a database. If itemset {a, b, c, d} is frequent, all its subset, {a, b, c}, {a, b, d}, {b, c, d}, {a, b}, {a, c}, {a, d}, {b, c}, {b, d}, {c, d}, {a}, {b}, {c}, and {d}, would be frequent because of this property. On the contrary, when Liu et al. (1999) employ the idea of multiple minimum supports, the *downward closure property* no longer holds. They use the *sorted closure property* to sort the items in an itemset according to their MIS values in ascending order to avoid the problem. For example, consider four items, a, b, c, and d, in a database. Their MIS values are MIS(a) = 10%, MIS(b) = 20%, MIS(c) = 5%, and MIS(d) = 6%. Assume we have an itemset {a, b} and its support is equal to 9%. The itemset will be infrequent because its support is smaller than its MIS value, i.e. MIS(a, b) = min[MIS(a), MIS(b)] = 10%. Then, itemsets {c, a, b} and {d, a, b} will not be generated. Notice that each item in the itemset is sorted by its MIS value. However, itemsets {c, a, b} and {d, a, b} still need to be generated because both MIS(c) = 5% and MIS(d) = 6% are smaller than 9%. Therefore, {c, a, b} and {d, a, b} become frequent even if their subset, {a, b}, does not.

Liu et al. (1999) modify the well-known Apriori algorithm so that the MSApriori algorithm can be used to find all frequent itemsets with multiple minimum supports. For further details, refer to the study by Liu et al. (1999). Lee et al. (2008a) proposed a fuzzy multiple-level mining algorithm with multiple minimum supports. Ouyang and Huang (2010) devised an algorithm for mining positive and negative sequential patterns with multiple minimum supports. Hu et al. (2010) developed a tree based approach to mining sequential pattern with multiple minimum supports. Summarily, these studies tackled the issue of multiple minimum supports but did not apply to mining periodic patterns.

### 2.2. The FP-tree and the FP-growth algorithm

Han et al. (2000) proposed a novel tree structure, called FP-tree, which is an extended prefix-tree structure for sorting compressed and crucial information. Consequently, the FP-growth method is a FP-tree-based mining algorithm for mining frequent patterns.

In the mining process, the frequent items only play a role before the construction of a FP-tree. All frequent items are sorted in non-increasing order of their support counts. The FP-tree consists of one root labeled as "*null*", a set of item prefix subtrees as the children of the root, and a frequent-item header table. Each node in the

item prefix subtree consists of three fields: *item-name*, *count*, and *node-link*. The field *count* registers the number of transactions represented by the portion of the path reaching this node, and *node-link* links to the next node in the FP-tree carrying the same item-name. Each entry in the frequent-item header table consists of two fields: *item-name* and *head of node-link*, which point to the first node in the FP-tree carrying the *item-name*.

A FP-tree is constructed as follows. Scan the database once to collect all frequent items and their supports. All frequent items are sorted in support nonincreasing order and denoted as *L*. After that, create the root of a FP-tree and label it as "null." Next, scan the database a second time to sort the items of each transaction in the database according to the order of *L*. On inserting a transaction, if the tree has the same path then the count of each node in the path is increased by 1. If the path is incomplete in the tree, then new branches and new nodes are created. And these new nodes' *node-link* will be linked to the nodes with the same *item-name* via the node-link structure. After constructing the FP-tree, the FP-growth algorithm recursively builds a *conditional pattern base* and a *conditional FP-tree* to generate all frequent patterns.

The essential difference between the PFP-tree and FP-tree is that the PFP-tree sets each item with different minimum supports, and uses the sorted closure property to append a node to those trees. Therefore, the construction of a PFP-tree is based on the support of each item in a real case and not only on one support for all items in a FP-tree.

### 2.3. The max-subpattern hit set algorithm

Han et al. (1999) presented several algorithms to efficiently mine partial periodic patterns. These algorithms explore some interesting properties related to partial periodicity, such as the Apriori and the max-subpattern hit set properties. In order to loosen the restrictions of the cyclic association rule, Han et al. (1999) used confidence to measure the level of significance of a periodic pattern. The confidence of a pattern is 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 a period of length 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 in any time series. As for the framework of mining association rules, a pattern is called a frequent partial periodic pattern in a time series if its confidence is larger than or equal to a threshold, *min_conf*. Therefore, the mining model is still applied to consider a single minimum support threshold.

A max-subpattern tree takes the candidate max-pattern, $C_{\max}$, as the root node. 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* may have a set of children if it contains more 2 non-* letters. Then the tree from the root of the tree is constructed and the missing non-* letters are checked in order to find the corresponding node. The count increases by 1 if the node *w* is found. Otherwise, a new node *w* (with count 1) and its missing ancestor nodes (only those on the path to *w*, with count 0) are created. If one exists, it (or them) is (or are) inserted into the corresponding place(s) of the tree. After a max-subpattern tree has been built, the tree is scanned to find frequency counts of the 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 the reachable ancestors. If the derived frequent pattern set is empty, then return. For more details, refer to the study by Han et al. (1999).

Previous studies of mining partial periodic patterns have been proposed in many applications. Ma and Hellerstein (2001) and Yang et al. (2004) proposed their algorithms to discover periodic patterns with noise, respectively. Cao et al. (2004) introduced a method to discover partial periodic patterns in discrete data sequences. Aref et al. (2004) developed an incremental and online algorithm for mining partial periodic patterns in time series databases. Huang and Chang (2005) extended the Yang et al. (2003) approach and devised SMCA to discover all patterns via two scans of temporal databases. Cao et al. (2007) discovered partial periodic patterns in spatiotemporal data. Lee et al. (2008) introduced fuzzy periodicity to mine fuzzy periodic association rules. Anwar et al. (2008) presented an efficient periodic patterns mining algorithm in post-mining environment. Gu and Dong (2009) proposed an algorithm to find the local frequent periodic patterns in time series data. Rasheed et al. (2011) devised an efficient algorithm using suffix trees to detect periodic patterns in time series database. In sum, these studies only dealt with the issue of mining periodic patterns without having the thought of multiple minimum supports.

## 3. PFP-tree: design and construction

In this section, we define the problem of partial periodic pattern mining and explore a method for PFP-tree construction. The tree structure can be used to find periodicity information efficiently.

### 3.1. Problem definition

Let $E = \{e_1, e_2, \ldots, e_m\}$ be a set of events, where $e_i$ denotes an independent event for $1 \le i \le m$. An event set $D$ is a nonempty subset of $E$, i.e. $D \subseteq E$. A sequence of event sets $S_D = \langle D_1, D_2, \ldots, D_n\rangle$ is a time series of events, where $D_j$ is an event set for $1 \le j \le n$, and $n$ is the number of event sets in $S_D$ (or called *length*). For brevity, the brackets can be omitted if $D_j$ has only one event.

**Example 1.** The length of sequence $S_D = \langle$a{b, c}baebaced$\rangle$ is 10. Event a occurs in $D_1$ and events b and c occur in $D_2$ simultaneously. The remaining event sets can be explained in the same way. Except event set {b, c}, we omit the remainders' brackets since each of them has only one event.

#### 3.1.1. Period segment

To divide a sequence $S_D$ into different periods, we develop a format of a segmented sequence, transformed from $S_D$. A segmented sequence $S$ is denoted as $S = \langle S_1, S_2, \ldots, S_m\rangle$ and its length can be divided into disjoint $m$ segments consisting of event sets. Let the period length of each segment be $l$. The form of $S_j$ $(1 \le j \le m)$, $\langle D_{(j-1)\times l+1}, D_{(j-1)\times l+2}, \ldots, D_{(j-1)\times l+l}\rangle$, is defined as a period segment (abbreviated as *ps*), where the subscript of $D$ indicates the ordinal position of $D$, counting from the beginning of the sequence $S$, and $m$ is the number of segments, i.e. $m = \lfloor n/l \rfloor$. Additionally, each event set in $S_j$ is decomposed into the different events successively. The *ps* is denoted as $S_j = \langle s_0, s_1, \ldots, s_{r-1}\rangle$, where $s_i$ $(0 \le i < r)$ is an event set and $r$ in the ordinal position of $S_j$, counting from $s_0$. Each event in $s_i$ can be represented as a tuple $(e_i d_i)$, where $e_i$ is the event and $d_i$ is the occurring position of $e_i$ in $S_j$ for $0 \le d_i \le l-1$. The tuple $(e_i d_i)$ is defined as an element of the period segment $S_j$, and a set of period segments is called a period segment database (abbreviated as *PSD*). The advantage of using the form of the tuple is that we can know directly when each event occurs in a sequence.

**Example 2.** Following Example 1 and setting a period length of 3 ($l = 3$), the sequence $S$ can be divided into three period segments ($m = \lfloor 10/3 \rfloor = 3$), of which lengths are all equal to 3, and the results are shown in the second column of Table 1. We say that the period segment $S_1 = \langle$a{b, c}b$\rangle$ is a subsequence of $S = \langle$a{b, c}baebaced$\rangle$. For event a in $S_1$, we know that it occurs at position 0 ($d = 0$) of the period segment $S_1$, i.e. $e_0$ = a, which can be denoted as element $s_0 = (e_0 d_0) =$ (a0). Then, the next position ($d = 1$) of event a in $S_1$

**Table 1**
The set of period segments *PSD*.

| $S_i$ | Period segment | Element set |
|---|---|---|
| $i=1$ | ⟨a{b, c}b⟩ | (a0), (b1), (c1), (b2) |
| $i=2$ | ⟨aeb⟩ | (a0), (e1), (b2) |
| $i=3$ | ⟨ace⟩ | (a0), (c1), (e2) |

has two events, b and c, denoted as elements $s_1=(e_1d_1)=$(b1) and $s_2=(e_2d_2)=$(c1), respectively. Finally, only event b occurs at position 2 ($d=2$) of the period segment $S_1$. So, event b can be denoted as element $s_3=(e_3d_3)=$(b2). The transformations of period segments $S_2$ and $S_3$ are the same as that of $S_1$. Afterward, we collect the *PSD* of the set of all period segments and present it in the last column of Table 1.

### 3.1.2. Periodic pattern

A pattern with period length $l$ is a nonempty element set $P$ over period segments. The pattern is a nonempty element subset of a period segment in *PSD*. The element length of pattern $P$ denotes the number of elements, and the pattern with $k$ length is called a *k-pattern*.

**Example 3.** A pattern $P=\{$(a0), (b2)$\}$ belongs to period length 3 ($l=3$), and its element length is 2. Thus, we call it a 2-pattern. According to the illustration in Example 2, we know that event a is at position 0 and event b is at position 2.

Moreover, we define that a nonempty pattern $P'$ is a subpattern of pattern $P$, if each element in $P'$ is also an element in $P$, i.e. $P' \subseteq P$.

**Example 4.** Suppose we have two patterns, $P=\{$(a0), (c1), (b2)$\}$ and $P'=\{$(a0), (b2)$\}$. We say that $P'$ is a subpattern of $P$ because the two elements, (a0) and (b2), in $P'$ are also in $P$.

### 3.1.3. The frequency count and support

A pattern $P$ can be matched with a period segment $S_i$ if $P$ is a subpattern of $S_i$. The frequency count (abbreviated as *count*) of pattern $P$ in *PSD*, *count*($P$), is the number of period segments matching pattern $P$. The support (abbreviated as *supp*) of pattern $P$ is defined as *supp*($P$) = *count*($P$)/$m$, where *count*($P$) is the number of period segments matching pattern $P$ and $m$ is the maximum number of periods of length contained in the time series. Following Example 2, we have a pattern $P=\{$(a0), (b2)$\}$ and its *count* and *supp* in the sequence are *count*($P$) = 2 and *supp*($P$) = 2/3 = 66% ($m=3$), respectively.

### 3.1.4. Multiple minimum supports

Each event in the time series databases is a pattern, and its minimum event support value (abbreviated as *MES*) has to be specified. The *MES* assignment for each event can be classified into two methods: (1) the automatic computation method (Liu et al., 1999) and (2) the manual method, i.e. all MESs are specified by users based on their domain knowledge. Since the experiments of the past study (Liu et al., 1999) were adopted by the first one, we employ it in our following experiments as well.

A pattern is called frequent if its support is greater than or equal to *MIN*. We describe the problem as follows. Let *MIN* be the smallest MES value of all events in $P=\{s_1, s_2, \ldots, s_k\}$ so that *MIN* is equal to min(MES($s_1$), MES($s_2$), ..., MES($s_k$)), and let *MIN_F* denote the set of those elements whose supports are no less than *MIN*. The elements in *MIN_F* are sorted in nonincreasing order according to their MES values.

**Example 5.** Results obtained in Example 2 are used to perform the following calculations. Four events are specified and their MES values are defined as: MES(a) = 100%, MES(b) = 100%, MES(c) = 66%, and MES(e) = 66%. Their frequency counts (supports) are shown in Table 2. Therefore, the value of *MIN* is equal to the minimum of the MESs or *MIN* = min(MES(a), MES(b), MES(c), MES(e)). If MES(a), MES(b), MES(c), and MES(e) are 100%, 100%, 66%, and 66%, respectively, *MIN* equals 66% and *MIN_F* = {(a0), (b2), (c1)}. Because the supports of b1, e1, and e2 are less than *MIN*, none of them are included in *MIN_F*.

**Lemma 1** (.). *Let $L_k$ denote the set of all frequent k-patterns, then each element of a pattern in $L_k$ must be in MIN_F.*

**Rationale.** Suppose that a $k$-pattern $P$ exists in $L_k$, where $P=\{s_1, s_2, \ldots, s_k\}$. If an element of $P$ doest not exist in *MIN_F*, then min(MES($s_1$), MES($s_2$), ..., MES($s_k$)) is less than *MIN*. However, *MIN* be the smallest MES value of all events, so min(MES($s_1$), MES($s_2$), ..., MES($s_k$)) does not exist. Thus, each element of a pattern in $L_k$ must be in *MIN_F*. □

For example, let a 3-pattern $P=\{$(a0),(b1),(c2)$\}$ exist in $L_3$. If $P$ is in $L_3$, the MES values of elements in $P$ are then not less than *MIN* so that all elements of $P$ are in the *MIN_F*. If the MES of any element in $P$ is less than *MIN*, $P$ does not exist in $L_3$.

The PFP-tree, therefore, consists of not only all frequent elements but also those infrequent elements with supports no less than *MIN*. According to Lemma 1, we must keep those elements which belong to *MIN_F*, because their supersets may be frequent.

**Example 6.** In Example 2, we know that *supp*(a0) = 100%, *supp*(b1) = 33%, *supp*(c1) = 66%, and *supp*(b2) = 66%, and the set of *MIN_F* = {(a0), (b2), (c1)}. In {(a0), (b2), (c1)}, the elements are sorted in nonincreasing order according to their MES values. Consider the small element (b2), where *supp*(b2) = 66% and MES(b) = 100%. We must retain it because any new patterns discovered in the following passes, such as {(b2), (c1)}, may be frequent. But if *supp*(b2) is less than *MIN*, we discard it directly.

## 3.2. PFP-tree construction

We propose a PFP-tree (Periodic FP-tree) for mining partial periodic patterns with multiple minimum supports. It is designed by modifying the FP-tree structure. We define the PFP-tree as follows.

**Definition 1** *(PFP-tree).* A PFP-tree is a tree structure defined as follows.

1. It consists of one root labeled as "*null*", a set of element prefix subtrees as the children of the root, and a frequent-element-header table which contains all elements in *MIN_F*.
2. Each node in the element prefix subtree consists of three fields: *element-name*, *count*, and *node-link*, where *element-name* registers which element the node represents, *count* registers the number of transactions represented by the portion of the path reaching the node, and *node-link* links to the null if there is none or to the next node in the PFP-tree carrying the same element-name.
3. Each entry in the frequent-element-header table consists of three fields, (1) *element-name*, (2) *MES*, and (3) *head of node-link*, which point to the first node in the PFP-tree carrying the *element-name*.
4. All the elements in the table are sorted in decreasing order in terms of their MES values. A node is arranged according to its element occurring position in increasing order if the node's *element-name* is the same as one with a different *element* occurring position.
5. If there is a node $y$ except the root in a PFP-tree and node $y$ is linked to node $x$, the path composed of some nodes from node $x$ to the root is called the *prefix subpath* of node $y$, the path composed of all nodes from node $x$ to the root is called the *prefix path* of node $y$ and those nodes in the path are called the *prefix nodes* of node $y$. On the contrary, if there is a node $z$ except any leaves in

**Table 2**
The frequency count (support) of each element in *PSD*.

| | Element | | | | | |
|---|---|---|---|---|---|---|
| | (a0) | (b1) | (c1) | (e1) | (b2) | (e2) |
| Count (support) | 3 (100%) | 1 (33%) | 2 (66%) | 1 (33%) | 2 (66%) | 1 (33%) |

a PFP-tree and node $z$ is linked to node $y$, the path composed of some nodes from node $z$ to the leaf is called the *postfix subpath* of node $y$, the path composed of all nodes from node $z$ to the leaf is called the *postfix path* of node $y$ and those nodes in the path are called the *postfix nodes* of node $y$.

Based on Definition 1, we devise the following PFP-tree construction algorithm and each function used in Algorithm 1 is shown in Fig. 1.

Fig. 1 gives the PFP-tree construction algorithm. Lines 1, 2, and 3 are the data preprocessing steps of the algorithm. Line 1 divides the time series database *TSD* into several period segments, each of which is equal to period length $l$ (**Algorithm 1**, Line 1.1). Then, we collect *PSD*, the set of all period segments (**Algorithm 1**, Line 1.2). Line 2 counts each element support to determine *MIN_F* (**Algorithm 1**, Line 2.1), the set of those elements with supports no less than *MIN* (**Algorithm 1**, Line 2.2). We discard the elements which do not belong to *MIN_F* in each period segment of *PSD* (**Algorithm 1**, Line 3.1). Then, we sort all elements according to their *MES* values in nonincreasing order (**Algorithm 1**, Line 3.2). At the end of the data preprocessing step, *PSD* will become the source database in the mining process. Line 4 consists of two steps (**Algorithm 1**, Lines 4.1 and 4.2). First, we create the root of a tree, labeled with "null". Second, we scan each period segment to construct the branch of the tree. Then, the procedure is called *insert_tree* in Fig. 2. If the tree has a node's name which is the same as its element-name, then we increase its count by 1 (**Procedure insert_tree**, Line 2). Otherwise, a new node is created and its count is set to 1. Moreover, its parent

**Algorithm 1 (The PFP-tree construction Algorithm)**

Input: a time series database *TSD*, a minimum support threshold of each event *MES*, and a period length $l$.

Output: PFP-tree.

Method:

1 Divide *TSD* into several period segments, each of which equals to period length $l$.
  1.1 Derive the elements in each period segment.
  1.2 Collect *PSD*, the set of all period segments, as a new database.
2 Scan *PSD* once.
  2.1 Calculate the support value of each element in *PSD*.
  2.2 Collect *MIN_F*, the set of those elements with supports no less than *MIN*.
3 Let $f$ denote an element in *MIN_F*. For each $f$ in *MIN_F* do the following:
  3.1 Delete the element in period segments with element$\notin$*MIN_F*.
  3.2 Sort all elements in period segments according to their *MES* values in nonincreasing order.
4 Create the root of a tree $T$, and label it as "null". For each period segment in *PSD* do the following:
  4.1 Let the sorted elements in period segments be $[p|P]$, where $p$ is the first element and $P$ is the remaining list.
  4.2 Call *insert_tree*($[p|P]$, $T$).
5 Name the resulting table as a frequent-element-header table.

**Fig. 1.** The PFP-tree construction algorithm.

**Table 3**
A time series database *TSD*.

| Time | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Events | b | c | a, c | e, f | b, d | c | a | d | a | d |
| Time | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
| Events | a, f | a | a | e | a, b | b | a | c, e | a | c |

```
Procedure insert_tree([p|P], T)
1   While (P is nonempty) {
2     If T has a child N such that N.element-name=p.element-name Then
        N.count ++ ;
3     Else
4       Create a new node N;
5       N.count=1;
6       Let its parent link be linked to T;
7       Let its node-link be linked to the nodes with the same element-name via
        the node-link structure;
8     End If
9   }
```

**Fig. 2.** The Procedure insert_tree for the PFP-tree construction.

link is linked to the tree, and its node-link is linked to the next node which has the same element-name (**Procedure insert_tree**, Lines 4–7). For traversing the PFP-tree, a frequent-element-header table is built (**Algorithm 1**, Line 5). We illustrate the PFP-tree construction algorithm with Example 7.

**Example 7** *(The construction of PFP-tree).* Let a time series database *TSD* be in Table 3. The period length is set to 4. The MES value and its corresponding frequency count of each event are shown in Table 4. At first, it is a part of data preprocessing (**Algorithm 1**, Lines 1–3). We scan *TSD* and separate it into period segments of the same period length, *PSD*, which is shown in the second column of Table 5. According to Lemma 1, we know that only those elements with support values no less than *MIN* will play a role in the mining process. The process consists of two phases. First, we scan *PSD* and derive the support of each element in Table 6. Second, we discard those elements whose support values are less than *MIN* in period segments. According to Algorithm 1, the order of the elements in the PFP-tree is arranged according to their MES values in nonincreasing order. The result of each period segment is listed in this order in the last column of Table 5. Table 6 shows the actual frequency counts (supports) of elements in *PSD*. We use element (a0) as an example. We count element (a0) if it appears in the period segments of *PSD* of Table 5. After traversing Table 5, we find that element (a0) appears in Segment IDs 3, 4, and 5. We know that the frequency count (support) of the element is equal to 3 (60%). The remaining elements for computing their frequency counts (supports) are shown in Table 6.

The algorithm needs to scan *PSD* twice to construct a PFP-tree. We first show the construction outcome of the PFP-tree in Fig. 3.

**Table 4**
The MES value and its corresponding frequency count of each event in *TSD*.

| | Event | | | | | |
|---|---|---|---|---|---|---|
| | a | b | c | d | e | f |
| MES value | 80% | 60% | 60% | 60% | 40% | 40% |
| Frequency count | 4 | 3 | 3 | 3 | 2 | 2 |

**Table 5**
A period segment database *PSD*.

| Segment ID | Period segment | (Ordered) frequent elements |
|---|---|---|
| 1 | (b0), (c1), (a2), (c2), (e3), (f3) | (a2), (b0), (c1) |
| 2 | (b0), (d0), (c1), (a2), (d3) | (a2), (b0), (c1) |
| 3 | (a0), (d1), (a2), (f2), (a3) | (a0), (a2) |
| 4 | (a0), (e1), (a2), (b2), (b3) | (a0), (a2), (e1) |
| 5 | (a0), (c1), (e1), (a2), (c3) | (a0), (a2), (c1), (e1) |

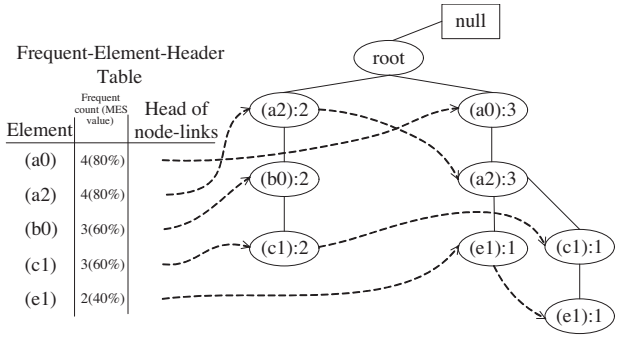


**Fig. 3.** The PFP-tree of Example 7.

**Table 6**
The frequency count (support) of each element in *PSD*.

| Element | a0 | a2 | a3 | b0 | b2 | b3 | c1 | c2 | c3 |
|---|---|---|---|---|---|---|---|---|---|
| Count (support) | 3 (60%) | 5 (100%) | 1 (20%) | 2 (40%) | 1 (20%) | 1 (20%) | 3 (60%) | 1 (20%) | 1 (20%) |
| Element | d0 | d1 | d3 | e1 | e3 | f2 | f3 | | |
| Count (support) | 1 (20%) | 1 (20%) | 1 (20%) | 2 (40%) | 1 (20%) | 1 (20%) | 1 (20%) | | |

The first scan of *PSD* retrieves a set of frequent elements. Then, the retrieved frequent elements are arranged by their MES values in nonincreasing order. In the second scan, to create the PFP-tree, we first create the root of a tree, labeled as "null". The scan of the first period segment in Table 5 leads to the construction of the first branch of the tree: ⟨(a2): 1, (b0): 1, (c1): 1⟩. Those frequent elements in all period segments are ordered according to their MES values in nonincreasing order. The second period segment {(a2), (b0), (c1)} is identical to the first one. The path is shared with the count of each node along the path incremented by 1, i.e. ⟨(a2): 2, (b0): 2, (c1): 2⟩. The scan of the third period segment leads to the construction of the second branch of the PFP-tree, ⟨(a0): 1, (a2): 1⟩. Next, for the fourth period segment {(a0), (a2), (e1)}, the first two elements are the same as the existing path ⟨(a0), (a2)⟩. Therefore, the count of each node along the path is incremented by 1, i.e. ⟨(a0): 2, (a2): 2⟩. For the last element (e1) of the fourth period segment, however, one new node ((e1): 1) is created and linked as the child of ((a2): 2). The last period segment, {(a0), (a2), (c1), (e1)}, shares a common prefix ⟨(a0), (a2)⟩ with the existing path ⟨(a0), (a2), (e1)⟩. The count of each node along the prefix is incremented by 1, i.e. ⟨(a0): 3, (a2): 3⟩. Then, for the remaining elements {(c1), (e1)}, a new path ⟨(c1): 1, (e1): 1⟩ is created and linked as a postfix path of ((a2): 3). After the algorithm scans all the period segments, the tree with the associated node-links is completed, as shown in Fig. 3.

In constructing the PFP-tree, the important property of the PFP-tree is that the PFP-tree contains the complete information for mining patterns.

**Lemma 2.** *Given a PSD and a support threshold MES for each element, the frequency count (support) of every frequent elements can be derived from PSD PFP-tree.*

**Rationale.** Based on the PFP-tree construction process, for each period segment in the *PSD*, its frequent element projection is mapped to one path in the PFP-tree. Given a frequent pattern $S = \langle s_1, s_2, \ldots, s_n \rangle$ in which elements are sorted in according to their MES values in nonincreasing order. Following the side-link of element $s_n$, we can visit all the nodes with label $s_n$ in the tree. For each path $p$ from the root to a node $v$ with label $s_n$, the count $s_n$: *count* in node $v$ is the number of transactions represented by $p$. If $\langle s_1, s_2, \ldots, s_n \rangle$ all appear in $p$, then the $s_n$: *count* transactions represented by $p$ contain $S$. Thus, we accumulate such counts. The sum is the count of $S$. □

In Example 7, we explain how to construct a PFP-tree. The algorithm employs the ordered frequent elements in the last column of Table 5 to construct a PFP-tree in Fig. 3. The process of constructing a PFP-tree describes Lemma 2.

**Lemma 3** (Space complexity of Algorithm 1). *Given a period segment database PSD and a minimum support threshold of each event MES, the number of nodes in an PFP-tree is no more than* $\sum_{ps \in PSD} |freq(ps)| + 1$, *where* $freq(ps)$ *is the set of frequent elements in ps. Moreover, the number of nodes in the longest path from the root is* $\max_{ps \in PSD} \{|freq(ps)|\}$.

**Rationale.** According to the PFP-tree construction process, for any period segment $ps$ in *PSD*, let $freq(ps) = s_1, s_2, \ldots, s_n$. A path exists, $\text{root} - s_1 - s_2 - \cdots - s_n$, in the PFP-tree. Except for when the root node that is empty and eliminated from the tree, all other nodes correspond to at least one frequent element occurring in the *PSD*. In the worst case, there is no overlap among frequent element projections of period segments; and thus all paths from the root to the leaves share only the root node. Therefore, the number of nodes in the tree is no more than $\sum_{ps \in PSD} |freq(ps)| + 1$. In the longest path from the root in the tree, there are $\max_{ps \in PSD} \{|freq(ps)|\}$ nodes. □

## 4. Mining frequent periodic patterns using a PFP-tree

In this section, we study how to explore the information stored in the PFP-tree, and develop an efficient PFP-growth algorithm for mining frequent partial periodic patterns. We observe two properties of the PFP-tree structure in Section 4.1 and introduce the PFP-growth algorithm in Section 4.2.

### 4.1. PFP-tree property

Here, explanations of prefix and postfix paths are proposed in the flowing paragraphs.

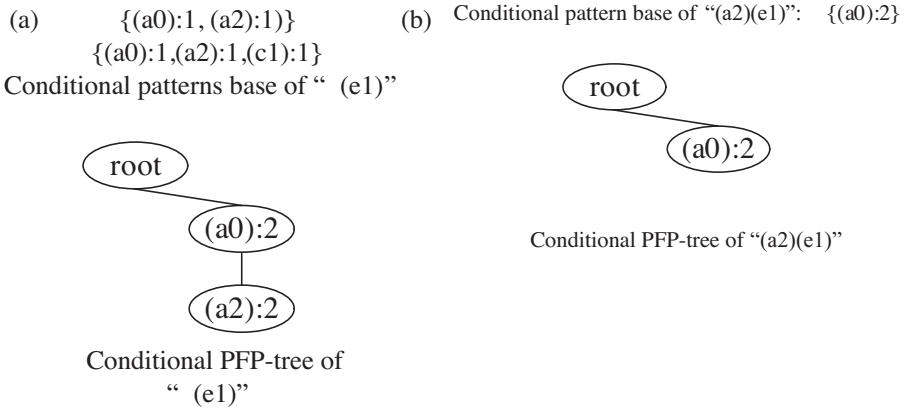


**Fig. 4.** (a) (e1)'s conditional PFP-tree. (b) (a2)(e1)'s conditional PFP-tree.

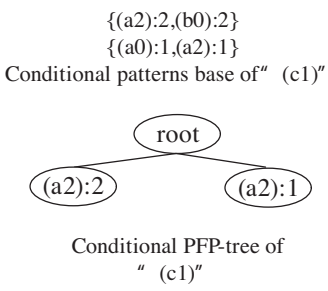


**Fig. 5.** (c1)'s conditional PFP-tree.

**Example 8.** The node (e1) in Fig. 3 derives two prefix paths: ⟨(a0), (a2), (e1)⟩ and ⟨(a0), (a2), (c1), (e1)⟩ in the PFP-tree. The two paths in the PFP-tree are terminated at their respective nodes, (e1)s. Therefore, node (e1) is a leaf node. The two paths, ⟨(a0), (a2)⟩ and ⟨(a0), (a2), (c1)⟩, form prefix paths of node (e1). The path ⟨(a0), (a2)⟩ also forms a prefix path of node (c1). Node (a0) is a prefix node of node (a2) since node (a0) links node (a2). Similarly, node (e1) forms the postfix path of node (a2), and it is linked to node (a2).

Two properties of the PFP-tree structure are introduced as follows.

**Property 1** (Node-link property). *For any frequent element $s_i$, all the possible frequent patterns containing only frequent elements and $s_i$ can be obtained by following $s_i$'s node-links, starting from the $s_i$'s head in the PFP-tree frequent-element-header.*

This property is based directly on the PFP-tree construction process. It allows us to find all of $s_i$'s pattern information through traversing the PFP-tree once following $s_i$'s node-links. For example, a frequent element (e1) in Fig. 3 is derived for node (e1) and has two paths, {(a0): 3, (a2): 3, (e1): 1} and {(a0): 3, (a2): 3, (c1): 1, (e1): 1} in the PFP tree. The two paths are obtained by following (e1)'s node-links, starting from the (e1)'s head in the frequent-element-header of the PFP-tree.

**Property 2** (Prefix path property). *To compute the $s_i$'s frequent patterns in a path P, only $s_i$'s prefix subpathes in the PFP-tree needs to be accumulated. The frequency count of every node in the prefix path should hold the same count as node $s_i$.*

**Rationale.** Let the nodes along the path $P$ be labeled as $s_1, s_2, \ldots, s_n$ in such an order that $s_1$ is the root of the prefix subtree. The element $s_n$ in $P$ is the leaf of the subtree. The element $s_i$ $(1 \le i \le n)$ is the node being referenced. Based on the process of construction of the PFP-tree presented in Algorithm 1, for each prefix node $s_k$ $(1 \le k \le i)$, the prefix subpath of the node $s_i$ in $P$ occurs together with $s_k$ exactly $s_i$: *count* times. Thus, every such prefix node should hold the same count as node $s_i$. Notice that a postfix node $s_m$ (for $i < m \le n$) along the same path also co-occurs with node $s_i$. However, the patterns with $s_m$ will be generated at the examination of the postfix node $s_m$, enclosing them here will lead to redundant generation of the patterns. Therefore, we only need to examine the $s_i$'s prefix subpath in $P$. □

For example, in Fig. 3, node (b0) is involved in a path, ⟨(a2): 2, (b0): 2, (c1): 2⟩, which is calculated for the frequent patterns containing node (b0) in this path. Only the prefix subpath of node (b0), ⟨(a2): 2⟩, needs to be extracted, and frequency count of every node in the prefix path should carry the same frequency count as node (b0). That is, the frequency count of the node in the prefix path should be adjusted to ⟨(a2): 2⟩.

Notice that the set of $s_i$'s prefix subpaths form a small database of patterns which co-occur with $s_i$. Such a database of patterns occurring with $s_i$ is called $s_i$'s conditional pattern base, and is denoted as "pattern base|$s_i$". Then one can compute all the frequent patterns co-occur with $s_i$ in this $s_i$-conditional pattern base by creating a small PFP-tree, called $s_i$'s conditional PFP-tree and denoted as "PFP-tree|$s_i$".

### 4.2. The PFP-growth algorithm

In our research, the PFP-growth algorithm recursively builds a conditional PFP-tree from the PFP-tree and a conditional pattern base for the mining frequent patterns. A detailed description of the procedure is given in Example 9.

**Example 9.** Let us examine the mining process based on the constructed PFP-tree shown in Fig. 3. According to Property 1, we collect all the patterns that node $s_i$ participates with by starting from $s_i$'s head (in the frequent-element-header table) and following $s_i$'s node-links. Here, we start from the bottom of the header table.

A frequent pattern ((e1): 2) is derived for node (e1) and it has two paths, ⟨(a0): 3, (a2): 3, (e1): 1⟩ and ⟨(a0): 3, (a2): 3, (c1): 1, (e1): 1⟩. The first path indicates that ⟨(a0), (a2), (e1)⟩ appears once in the database. Notice the path also indicates that ⟨(a0), (a2)⟩ appears three times. Based on Property 2, we exclude the node "(e1)" itself and add the rest of the nodes to the conditional pattern base. Those nodes whose supports are not smaller than *MIN* are added the conditional PFP-tree. And the frequency count of each node should hold the same count as that of node (e1). So, we discard the node (e1) in the two paths. Two (e1)'s prefix paths, ⟨(a0): 1, (a2): 1⟩ and ⟨(a0): 1, (a2): 1, (c1): 1⟩, form (e1)'s conditional pattern base. Then, we use (e1)'s conditional pattern base to generate a (e1)'s conditional PFP-tree. We know that node (e1) shares a common prefix ⟨(a0), (a2)⟩ with the two paths. Therefore, {(a0), (a2)} appears twice together with node (e1). Taking note of node (c1), it only appears once in path with node (e1) and is discarded. After adding these paths ⟨(a0):

**Table 7**
Conditional pattern base and conditional PFP-tree.

| Element | MES | Conditional pattern base | Conditional PFP-tree |
|---|---|---|---|
| (e1) | 2 | {(a0): 1, (a2): 1}, {(a0): 1, (a2): 1, (c1): 1} | {(a0): 2, (a2): 2}\|(e1) |
| (c1) | 3 | {(a2): 2, (b0): 2}, {(a0): 1, (a2): 1} | {(a2): 3 }\|(c1) |
| (b0) | 3 | {(a2): 2} | ∅ |
| (a2) | 4 | {(a0): 3} | ∅ |
| (a0) | 4 | ∅ | ∅ |

**Table 8**
Conditional patterns and conditional frequent patterns.

| Element | Conditional patterns | Conditional frequent patterns | The Han et al. frequent pattern format |
|---|---|---|---|
| (e1) | {(a0), (e1)}, {(a2), (e1)}, {(a0), (a2), (e1)} | {(a0), (e1)}, {(a2), (e1)}, {(a0), (a2), (e1)} | {ae**}, {*ea*}, {aea*} |
| (c1) | {(a2), (c1)} | {(a2), (c1)} | {*ca*} |

**Algorithm 2 (PFP-growth for mining partial periodic pattern with multiple minimum supports)**

Input: PFP-tree and MES($s_i$) for each element $s_i$.

Output: The complete set of all $s_i$'s conditional frequent patterns and the set of all support values of $s_i$'s conditional patterns.

Method: call PFP-growth (PFP-tree, *null*, MES($s_i$)).

Procedure PFP-growth (*Tree*, $\alpha$, MES($\alpha$)) {

1 For each $s_i$ in the header table of *Tree* do {

2 Generate pattern $\beta = s_i \cup \alpha$ with support=$s_i$.support;

3 Construct $\beta$'s conditional pattern base and $\beta$'s conditional PFP-tree $Tree_\beta$;

4 If $Tree_\beta \neq \varnothing$ Then call PFP-growth ($Tree_\beta$, $\beta$, MES($\alpha$));

5 }

}

**Fig. 6.** The PFP-growth algorithm.

2, (a2): 2⟩, (e1)'s conditional PFP-tree is shown in Fig. 4a. We can obtain 2 elements through (e1)'s conditional PFP-tree. Then, we need to check the count of each element whether exceeds the corresponding count of the MES value for element (e1). If the former is greater than the latter, it is frequent. Otherwise, it is infrequent and then we need to remove it. We follow the node-link of each element and sum up the counts. For (e1)'s conditional PFP-tree, the count of element (a0) is 2 and (a2) is 2. Since the corresponding count of the MES value for element (e1) is 2, elements (a0) and (a2) are both frequent. We find that (e1) has two frequent patterns: {(a0): 2, (e1): 2} and {(a2): 2, (e1): 2}. After finding all of (e1)'s frequent patterns {(a0): 2, (e1): 2} and {(a2): 2, (e1): 2}, we respectively build ⟨(a0): 2, (e1): 2⟩ and ⟨(a2): 2, (e1): 2⟩ paths in element (e1)'s conditional PFP-tree. ⟨(a0): 2, (e1): 2⟩'s conditional pattern base contains no elements and would be terminated. For ⟨(a2): 2, (e1): 2⟩'s conditional pattern base and conditional PFP-tree, we find a frequent pattern, {(a0): 2, (a2): 2, (e1): 2}. The conditional PFP-tree of ⟨(a2): 2, (e1): 2⟩ is shown in Fig. 4b. We find two patterns {(a0): 2, (c1): 2, (e1): 2} and {(a2): 2, (c1): 2, (e1): 2}. In summary, we find that all of (e1)'s conditional patterns are {(a0): 2, (e1): 2}, {(a2): 2, (e1): 2} and {(a0): 2, (a2): 2, (e1): 2}. Also, the frequent periodic patterns are: {(a0): 2, (e1): 2}, {(a2): 2, (e1): 2}, and {(a0): 2, (a2): 2, (e1): 2}. The PFP-growth algorithm for element (e1) will be terminated if (e1)'s conditional pattern base is null.

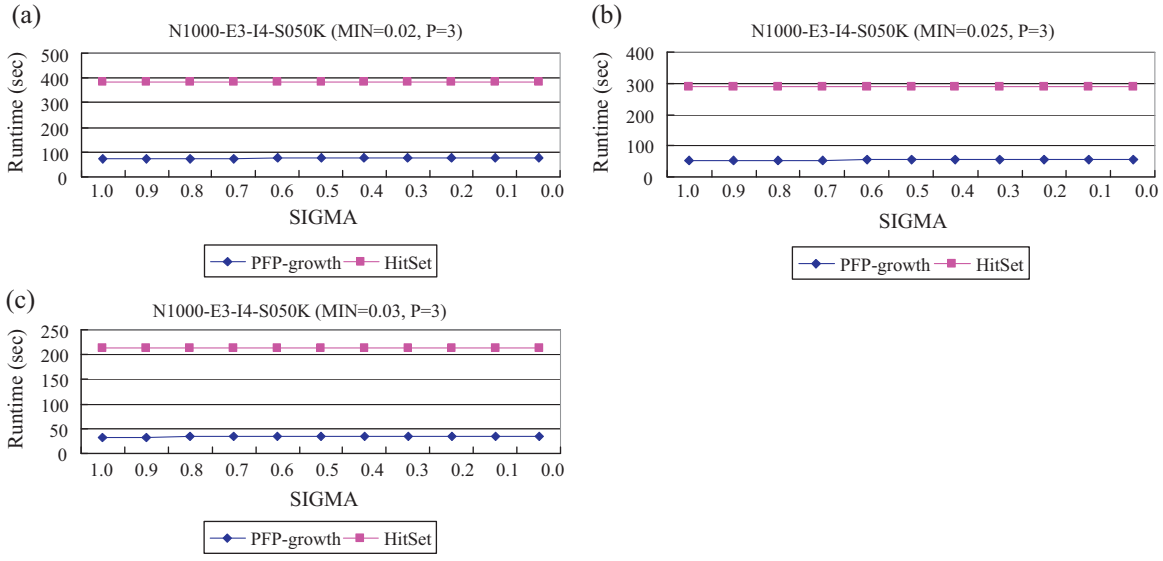


**Fig. 7.** (a) Run times for dataset N1000-E3-I4-S050K (*MIN* = 2%, *P* = 3). (b) Run times for dataset N1000-E3-I4-S050K (*MIN* = 2.5%, *P* = 3). (c) Run times for dataset N1000-E3-I4-S050K (*MIN* = 3%, *P* = 3).

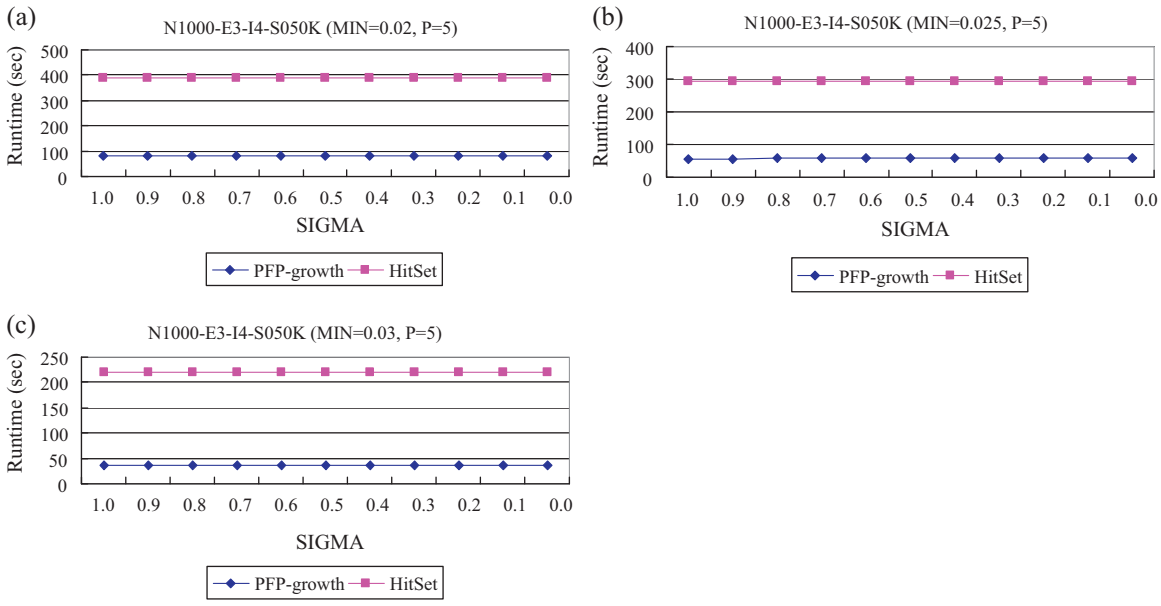


**Fig. 8.** (a) Run times for dataset N1000-E3-I4-S050K (*MIN* = 2%, *P* = 5). (b) Run times for dataset N1000-E3-I4-S050K (*MIN* = 2.5%, *P* = 5). (c) Run times for dataset N1000-E3-I4-S050K (*MIN* = 3%, *P* = 5).

Similarly, node (c1) has only one path, ⟨(a2): 3, (c1): 3⟩. Then, we exclude the node (c1) in the path, leaving ⟨(a2): 3⟩. The element (c1)'s conditional PFP-tree is shown in Fig. 5. After finding the (c1)'s conditional pattern base, we build (c1)'s conditional PFP-tree, ⟨(a2): 3⟩. The pattern {(a2): 3, (c1): 3} in (c1)'s conditional pattern base is frequent, since its count is greater than the corresponding count of the MES value for element (c1). In summary, only one frequent periodic pattern, {(a2): 3, (c1): 3}, can be found.

Since the counts of elements in the (c1)'s conditional pattern bases, {(b0): 2} and {(a2): 2}, are less than the corresponding counts of their MES values, its conditional PFP-tree is not generated. Furthermore, the last node (a0) has no elements in its conditional pattern base, so its conditional PFP-tree is also not generated. Finally, all of the conditional pattern bases and the conditional PFP-trees are summarized in Table 7. Table 8 shows all conditional patterns, frequent patterns, and the Han et al. frequent pattern format (the conversion from our pattern format to theirs).

The developments of the FP-growth and PFP-growth algorithms both find frequent patterns based on the conditional FP-tree concept. However, their major difference is that the latter adopts a *sorted downward closure property* to prune unnecessary conditional patterns. Based on this property, all of the minimum supports of the elements in the conditional PFP-tree will not be less than the minimum support of the condition element. In other words, if an element is not frequent in a conditional PFP-tee, there is no longer any need to generate a condition PFP-tree.

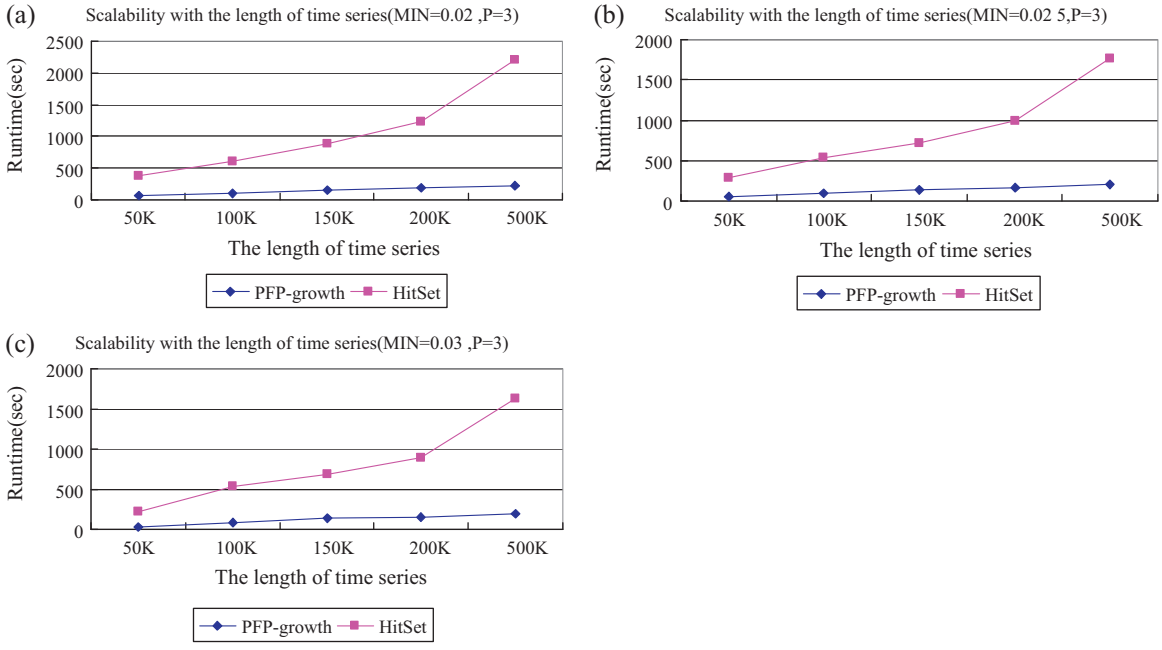


**Fig. 9.** (a) Scalability with the length of time series (*MIN* = 2%, *P* = 3). (b) Scalability with the length of time series (*MIN* = 2.5%, *P* = 3). (c) Scalability with the length of time series (*MIN* = 3%, *P* = 3).

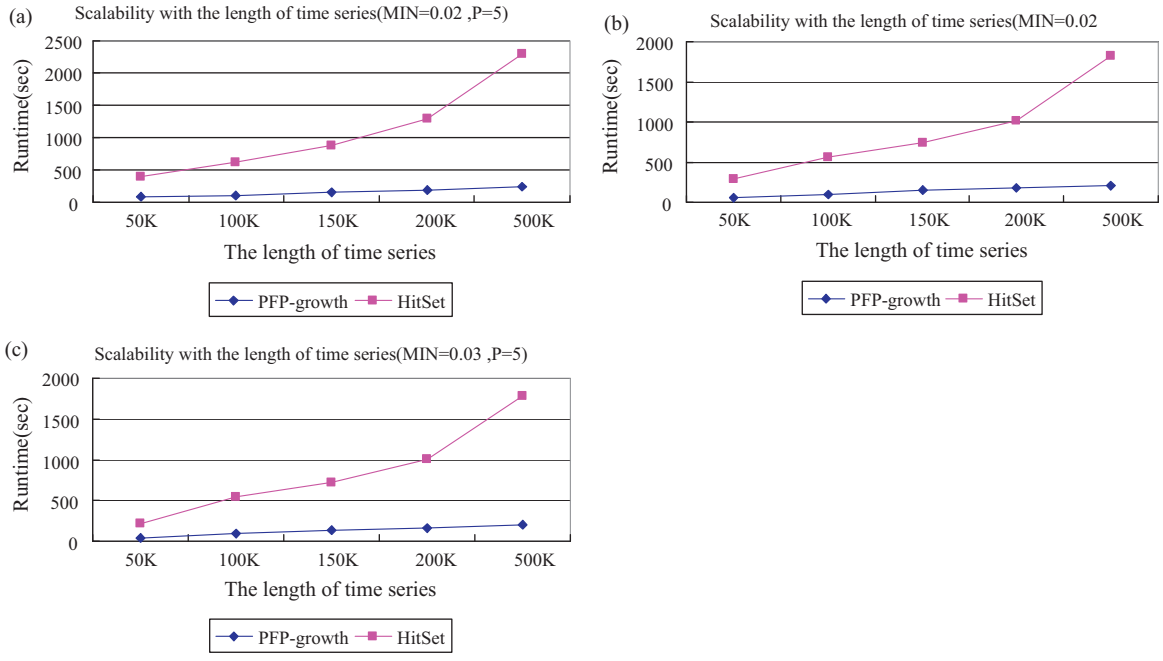


**Fig. 10.** (a) Scalability with the length of time series (*MIN* = 2%, *P* = 5). (b) Scalability with the length of time series (*MIN* = 2.5%, *P* = 5). (c) Scalability with the length of time series (*MIN* = 3%, *P* = 5).

The following lemma and corollary are related to our mining process.

**Lemma 4** (Fragment growth). *Let $\alpha$ be an element in PSD, B be $\alpha$'s conditional pattern base, and $\beta$ be an element in B. Then the frequency count (support) of $\alpha \cup \beta$ in PSD is equivalent to the frequency count (support) of $\beta$ in B.*

**Rationale.** According to the definition of conditional pattern base in Section 4.1, each period segment in *B* occurs under the condition of the occurrence of $\alpha$ in the *PSD*. If an element $\beta$ appears in *B t* times, it appears with $\alpha$ in *PSD t* times as well. Moreover, since all such elements are collected in $\alpha$'s conditional pattern base, $\alpha \cup \beta$ occurs exactly *t* times in *PSD* as well. Thus the lemma holds. □

In Example 9, element (e1)'s conditional pattern base involves element (a2). The frequency count of {(a2), (e1)} in *PSD* is 2 which is equal to that of element (a2) in element (e1)'s conditional pattern base.

**Corollary 1** (Pattern growth). *Let $\alpha$ be a frequent element in PSD, B be $\alpha$'s conditional pattern base, and $\beta$ be an element in B. Then $\alpha \cup \beta$ is frequent in PSD if and only if $\beta$ is frequent in B.*

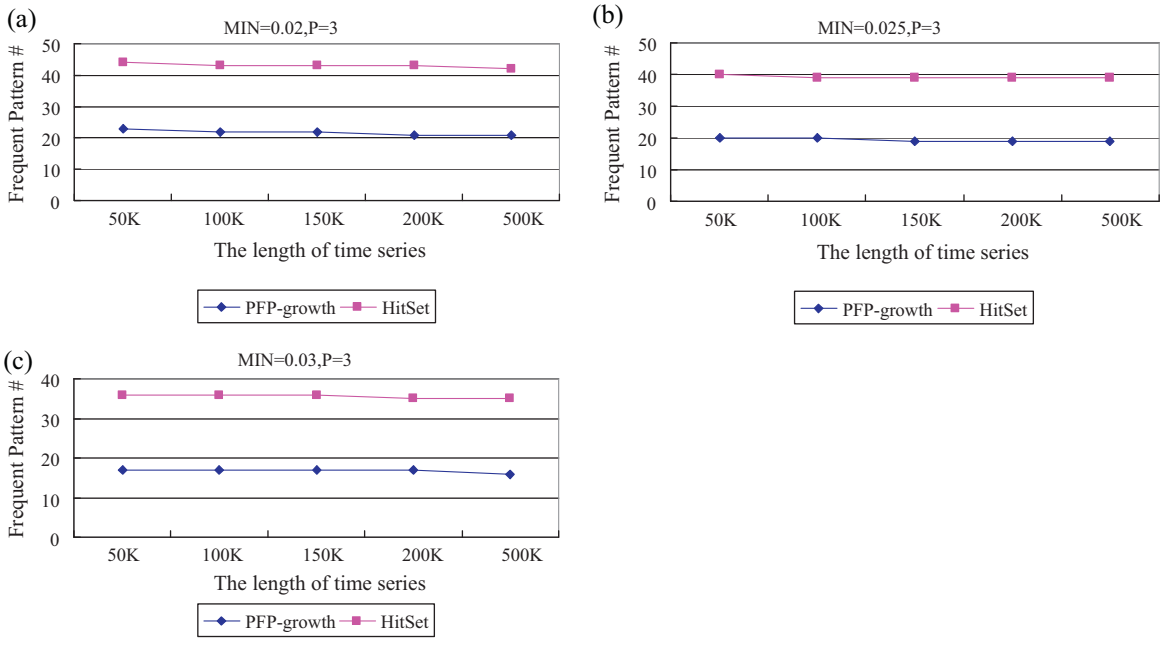


**Fig. 11.** (a) Frequent pattern # with length of time series (*MIN* = 2%, *P* = 3). (b) Frequent pattern # with length of time series (*MIN* = 2.5%, *P* = 3). (c) Frequent pattern # with length of time series (*MIN* = 3%, *P* = 3).

**Table 9**
Parameters.

| | |
|---|---|
| $\|S\|$ | The length of time series |
| $\|E\|$ | Average number of the event sets |
| $\|I\|$ | Average size of maximal potentially frequent patterns |
| $\|L\|$ | Number of maximal potentially frequent patterns |
| $P$ | A period length |
| $N$ | Number of events |

**Rationale.** This corollary is the case when $\alpha$ is a frequent element in *PSD*, and when the support of $\beta$ in $\alpha$'s conditional pattern base $B$ is no less than *MIN*. We first prove the "if" part. Suppose $\beta$ is frequent in $B$, that is, $\beta$ appears in $B$ at least $MIN \times m$ times ($m$ is defined in Section 3.1.3). Since $B$ is $\alpha$'s conditional pattern base, each period segment in $B$ appears under the existence of $\alpha$. That is, $\beta$ appears together with $\alpha$ in *PSD* at least $MIN \times m$ times. Therefore, $\alpha \cup \beta$ is frequent in *PSD*. Then, we prove the "only if" part. Suppose $\beta$ is not frequent in $B$. that is, $\beta$ appears in $B$ less than $MIN \times m$ times. Since $B$ is $\alpha$'s conditional pattern base, all the elements containing $\beta$ and co-occurring with $\alpha$ are in $B$. Thus $\beta$ co-occurs with $\alpha$ less than $MIN \times m$ times. Therefore, $\alpha \cup \beta$ is not frequent in *PSD*. □

In Example 9, element (e1) is a frequent element in *PSD*, and its conditional pattern base involves element (a2). Since element (a2) is frequent, {(a2), (e1)} is also frequent.

Lemma 4 and Corollary 1 are similar to those proposed by Han et al. (2000). Therefore, we have the following algorithm for mining partial periodic patterns with multiple minimum supports. The PFP-growth algorithm (**Algorithm 2**) is shown in Fig. 8.

As shown in Fig. 6, we describe the PFP-growth algorithm briefly. At first, we construct a conditional pattern base and mine its conditional PFP-tree for each frequent element $s_i$. Therefore, we have the procedures of Lines 2 and 3. Otherwise, the process of mining frequent partial periodic patterns is then recursively executed on the pattern base $\beta$ by constructing a conditional PFP-tree and a conditional pattern base for $\beta$. Therefore, we have the procedure of Line 4. To streamline the presentation, we leave the analyses of correctness (Theorem 1) and completeness (Theorem 2) in Appendix A.

## 5. Experimental evaluation

In this section, we report an experimental study which compares the performances of the PFP-growth algorithm and the max-subpattern hit set algorithm (Han et al., 1999). These algorithms were implemented using Sun Java language (J2SDK 1.4.2_16) and tested on a PC with an Intel Core 2 Duo 2.4 GHz processor and 2GB main memory using the Windows Server 2003 operating system. Neither multithreading technology nor parallel computing skills were used in implementing our programs.

The synthetic datasets used for our experiments were generated by applying the standard procedure described in Han et al. (1999). Table 9 lists the parameters used in this simulation. We generate datasets by fixing $N = 1000$, $|E| = 3$, $|I| = 4$, $|L| = 2000$, and $|S| = 50$ K in all experiments. Moreover, the period length is assigned to 3 and 5.

In our experiments, we need a method to assign MES values to events in *PSD*. Therefore, we refer to the method proposed by Liu et al. (1999) to use the actual frequencies of the events in the *PSD* as the basis for the MES assignments. The formula of assigning MES values is $\text{MES}(e) = \max\{\sigma f(e), MIN\}$, where $f(e)$ is the actual support value of event $e$ in *PSD*, and $\sigma$ ($0 \le \sigma \le 1$) is a non-negative parameter, controlling that how the MES values for events are related to their support's values. If $\sigma = 0$, we have only one minimum support, *MIN*, which is the same as the traditional periodic pattern mining. On the contrary, if $\sigma = 1$ and $MIN \le f(e)$, $f(e)$ is the MES value for event $e$.

**Example 10.** Consider three events, a, b, and c in a time series dataset, where $f(a) = 1\%$, $f(b) = 3\%$, and $f(c) = 10\%$. If we set $MIN = 1\%$ and $\sigma = 0.3$, then $\text{MES}(a) = \max\{0.3 \times 1\%, 1\%\} = 1\%$, $\text{MES}(b) = \max\{0.3 \times 3\%, 1\%\} = 1\%$, and $\text{MES}(c) = \max\{0.3 \times 10\%, 1\%\} = 3\%$.

The first comparison considers the run times of two algorithms with different *MIN* values, 2%, 2.5%, and 3%. Then, we assign a period length of 3 ($P = 3$) for dataset N1000-E3-I4-S050K. The N1000-E3-I4-S050K means that it is generated with 1000 events, 3 events per event set on average, the average size of maximal potentially frequent patterns is 4, and the length of the time series is 50000. We show the results in Fig. 7(a) for $MIN = 2\%$ and $P = 3$, Fig. 7(b) for $MIN = 2.5\%$ and $P = 3$, and Fig. 7(c) for $MIN = 3\%$ and $P = 3$. The results all indicate that the PFP-growth algorithm is superior to the max-subpattern hit set algorithm (the compared baseline is observed when the PFP-growth algorithm's sigma is equal to 0.0). To find the reason, let us first note that although the max-subpattern hit set algorithm requires nothing but two scans of the database, it still needs to construct more possible infrequent patterns in the hit set with a max-subpattern tree structure. Whether a pattern is frequent or not in the tree and exerting the candidate max-pattern to span the following subpatterns will all worsen the performance of the max-subpattern hit set algorithm. According to our idea, however, we build a compressed data structure, a PFP-tree, to hold the entire time series database in the memory. Then, a divide-and-conquer strategy is used to find all frequent patterns. No candidate patterns will be generated. This explains why the performance of the PFP-growth algorithm is better than that of the max-subpattern hit set algorithm.

After testing the run times under different *MIN* values for $P = 3$, we then change the period length into 5 ($P = 5$) to repeat the above experiments in Fig. 11(a)–(c). We show the run times of dataset N1000-E3-I4-S050K for each *MIN* value. All results are observed in Fig. 8(a) for $MIN = 2\%$ and $P = 5$, Fig. 8(b) for $MIN = 2.5\%$ and $P = 5$, and Fig. 8(c) for $MIN = 3\%$ and $P = 5$. The results all demonstrate again that the performance of the PFP-growth algorithm is superior to that of the max-subpattern hit set algorithm. Besides, modifying the period length does not have much impact on the performance of the PFP-growth algorithm either.

Next, we study the scalabilities of the two algorithms. The tests are performed by using five datasets: N1000-E3-I4-D50K, N1000-E3-I4-D0100K, N1000-E3-I4-D150K, N1000-E3-I4-D200K, and N1000-E3-I4-D500K, with the *MIN* values varied from 2% to 3%. The arguments are executed for $P = 3$ and 5, respectively. Fig. 9(a)–(c) shows the results for $MIN = 2\%$ and $P = 3$, $MIN = 2.5\%$ and $P = 3$, and $MIN = 3\%$ and $P = 3$, and Fig. 10(a)–(c) shows the results for $MIN = 2\%$ and $P = 5$, $MIN = 2.5\%$ and $P = 5$, and $MIN = 3\%$ and $P = 5$. The reported run time is the average of the 10 tests for sigma from 0.1 to 1.0. All results show that the PFP-growth algorithm presents linear scalability with the length of time series from 50K to 500K, *MIN* from 2% to 3%, and $P$ from 3 to 5; however, the max-subpattern hit set algorithm does not. This experiment indicates that the PFP-tree algorithm is more scalable than the max-subpattern hit set algorithm.

According to the scale-up experiments, we study the number of frequent patterns generated by the two algorithms. Fig. 11(a)–(c) shows the results for $MIN = 2\%$ and $P = 3$, $MIN = 2.5\%$ and $P = 3$, and $MIN = 3\%$ and $P = 3$, and Fig. 12(a)–(c) shows the results for $MIN = 2\%$ and $P = 5$, $MIN = 2.5\%$ and $P = 5$, and $MIN = 3\%$ and $P = 5$. All the results show when the support threshold is decreased and when the number of frequent patterns is increased by both algorithms. However, the PFP-growth algorithm presents reasonable outcomes. A possible reason is that the max-subpattern hit set algorithm only considers one single minimum support to mine patterns in time series databases. The PFP-growth algorithm, however, uses

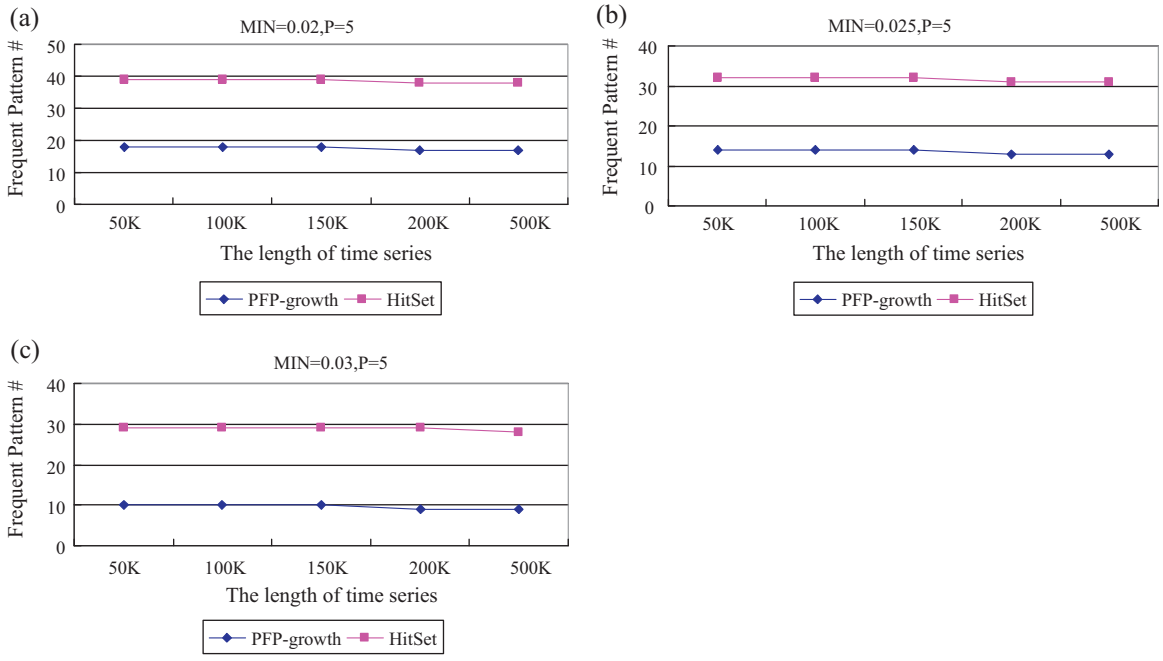


**Fig. 12.** (a) Frequent pattern # with length of time series (*MIN* = 2%, *P* = 5). (b) Frequent pattern # with length of time series (*MIN* = 2.5%, *P* = 5). (c) Frequent pattern # with length of time series (*MIN* = 3%, *P* = 5).

multiple minimum supports for the different frequencies of different events. This can certainly reduce the number of the possible redundant patterns which are found by the max-subpattern hit set algorithm. The results demonstrate our argument that some redundant patterns may not be available for realistic applications. Therefore, all the above experiments not only demonstrate our model's computational efficiency and scalability, but also prove the effectiveness of the patterns.

## 6. Conclusion and future work

The study explored the problem of mining partial periodicity in time series databases. Information about variations of events aids managers in identifying better trends in partial periodicity and attaining a better understanding of forecasting. Disadvantages of using the Apriori-property (Agrawal and Srikant, 1994) to discover periodic patterns can be eliminated. An efficient mining method, the PFP-growth algorithm, has also been developed. Results of performance analyses show that the proposed PFP-growth algorithm is more efficient and scalable than the max-subpattern hit set algorithm. In particular, when the MIN value is small or the scalability is extended, the PFP-growth algorithm outperforms the max-subpattern hit set algorithm. Based on the results of the number of frequent periodic patterns, mining patterns with multiple minimum supports is more realistic in real-life applications to assist the user in reducing the number redundant patterns for decision making.

The proposed algorithm can be applied to predict stock market price movement, shopping habits, and so on. For example, periodic information of customers' shopping behavior based on login time and searches for products could be available from a database maintained by an online shopping system. The periodicities for different event frequencies may reveal pertinent information for prediction and analysis. Therefore, different and specific information (e.g., new product information and on sales product information) can be made available to each customer (customization).

This study has confirmed the validity of using the proposed mining partial periodic patterns in a time series with a single level. Additionally, the model can also be extended to find partial periodicity with multiple minimum supports in multiple-level categorical data (such as: hour, day, and week). The algorithm and results are of great importance for understanding more general and linguistic information in order to make efficient decisions.

## Acknowledgments

The authors would like to thank the Area Editor, Dr. K. Dutta, and anonymous reviewers for their helps and valuable comments to improve this paper. This research was supported by the National Science Council of the Republic of China under the grant NSC 99-2410-H-194-063-MY2.

## Appendix A.

**Theorem 1.** *The patterns obtained by the PFP-growth algorithm are correct.*

**Rationale.** Since Line 2.2 in Algorithm 1 removes elements whose supports are less than *MIN*, each possible frequent pattern of length 1 can be collected in *MIN_F*. After deleting the elements of the period segments that do not exist in *MIN_F*, Line 4 in Algorithm 1 starts to construct the PFP-tree. Assume that every pattern of length $k$ obtained by Algorithm 2 is frequent, i.e. $\alpha$ is frequent. Because of the sorted downward closure property, $\beta$ obtained by $s_i \cup \alpha$ is also frequent. This proves the induction. □

**Theorem 2.** *The PFP-growth algorithm can find every frequent pattern.*

**Rationale.** Based on the PFP-tree construction process (Algorithm 1), for each element in the *PSD*, its possible frequent element projection is mapped to a path from the root in the PFP-tree.

Given a frequent pattern $S = \langle s_1, s_2, \ldots, s_n \rangle$ in which elements are sorted in the support descending order, by following the side-link of element $s_n$, we can visit all the nodes with label $s_n$ in the tree.

For each path $p$ from the root to a node $r$ with label $s_n$, the frequent count $count(r)$ in node $r$ is the number of period segments

represented by $p$. If $s_1, s_2, \ldots, s_n$ all appear in $p$, then the $count(r)$ period segments represented by $p$ contain $S$. Therefore, we accumulate the frequent counts. The sum is the frequent count of $S$.

After a PFP-tree is constructed, it contains complete information for mining frequent patterns from the *PSD*. Thus, the PFP-growth algorithm can find all frequent patterns in this tree. □

## References


Agrawal, R., Srikant, R., 1994. Fast algorithms for mining association rules in large databases. In: Proceedings of the 20th International Conference on Very Large Data Bases , pp. 487–499.

Aref, W.G., Elfeky, M.G., Elmagarmid, A.K., 2004. Incremental, online, and merge mining of partial periodic patterns in time-series databases. IEEE Transactions on Knowledge and Data Engineering 16 (3), 332–342.

Anwar, F., Petrounias, I., Kodogiannis, V.S., Tasseva, V., Peneva, D., 2008. Efficient periodicity mining of sequential patterns in a post-mining environment. In: 2008 4th International IEEE Conference "Intelligent Systems" , pp. 16-2–16-11.

Cao, H., Cheung, D.W., Mamoulis, N., 2004. Discovering partial periodic patterns in discrete data sequences. In: Proceedings of the 8th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining , pp. 653–658.

Cao, H., Mamoulis, N., Cheung, D.W., 2007. Discovery of periodic patterns in spatiotemporal sequences. IEEE Transactions on Knowledge and Data Engineering 19 (4), 453–467.

Gu, C.-K., Dong, X.-L., 2009. Efficient mining of local frequent periodic patterns in time series database. In: 2009 International Conference on Machine Learning and Cybernetics , pp. 183–186.

Han, J., Dong, G., Yin, Y., 1998. Mining segment-wise periodic patterns in time-related databases. In: Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining , pp. 214–218.

Han, J., Dong, G., Yin, Y., 1999. Efficient mining of partial periodic patterns in time series database. In: Proceedings of the 15th International Conference on Data Engineering , pp. 106–115.

Han, J., Pei, J., Yin, Y., 2000. Mining frequent patterns without candidate generation. In: Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data , pp. 1–12.

Hu, Y.-H., Wu, F., Liao, Y.-C., 2010. Sequential pattern mining with multiple minimum supports: a tree based approach. In: International Conference on Software Engineering and Data Mining (SEDM) , pp. 428–433.

Huang, K.-Y., Chang, C.-H., 2005. SMCA: a general model for mining asynchronous periodic patterns in temporal databases. IEEE Transactions on Knowledge and Data Engineering 17 (6), 774–785.

Lee, W.-J., Jiang, J.-Y., Lee, S.-J., 2008. Mining fuzzy periodic association rules. Data & Knowledge Engineering 65, 442–462.

Lee, Y.-C., Hong, T.-P., Wang, T.-C., 2008a. Multi-level fuzzy mining with multiple minimum supports. Expert Systems with Applications 34, 459–468.

Liu, B., Hsu, W., Ma, Y., 1999. Mining association rules with multiple minimum supports. In: Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining , pp. 337–341.

Ma, S., Hellerstein, J.L., 2001. Mining partially periodic event patterns with unknown periods. In: Proceedings of the 17th International Conference on Data Engineering , pp. 205–214.

Ouyang, W., Huang, Q., 2010. Mining positive and negative sequential patterns with multiple minimum supports in large transaction databases. In: 2010 Second WRI Global Congress on Intelligent Systems , pp. 190–193.

Rasheed, F., Alshalalfa, M., Alhajj, R., 2011. Efficient periodicity mining in time series databases using suffix trees. IEEE Transactions on Knowledge and Data Engineering 23 (1), 79–94.

Yang, J., Wang, W., Yu, P.S., 2001. Infominer: mining surprising periodic patterns. In: Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining , pp. 395–400.

Yang, J., Wang, W., Yu, P.S., 2003. Mining asynchronous periodic patterns in time series data. IEEE Transaction on Knowledge and Data Engineering 15 (3), 613–628.

Yang, J., Wang, W., Yu, P.S., 2004. Discovering high order periodic patterns. Knowledge and Information Systems 6 (3), 243–268.



**Shih-Sheng Chen** received his Ph.D. degree in Information Management from National Central University of Taiwan in 2003. He is an Assistant Professor in the Department of Information Management at National Chin-Yi University of Technology. His current research interests include data mining in soft computing, decision support systems, information management, customer relationship management and business.

**Tony, Cheng-Kui Huang** received the Ph.D. degree in Information Management from National Central University of Taiwan in 2006. He is an Assistant Professor in the Department of Business Administration at National Chung Cheng University of Taiwan. His current research interests include data mining in business, decision support systems, information management, and soft computing. He has published papers in IEEE Transactions on Systems, Man and Cybernetics, Part B, Information Sciences, Data & Knowledge Engineering, Fuzzy Sets and Systems, Expert Systems with Applications, and Computers and Education.

**Zhe-Min Lin** received his MS degree in Information Management from Tatung University of Taiwan. His research interests are in data mining and information management.