Discovering Stable Periodic-Frequent Patterns in Transactional Data

Philippe Fournier-Viger[1(✉)], Peng Yang[2], Jerry Chun-Wei Lin[3], and Rage Uday Kiran[4]

[1] School of Humanities and Social Sciences, Harbin Institute of Technology (Shenzhen), Shenzhen, China
philfv8@yahoo.com
[2] School of Computer Science and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen, China
pengyeung@163.com
[3] Department of Computing, Mathematics and Physics, Western Norway University of Applied Sciences (HVL), Bergen, Norway
jerrylin@ieee.org
[4] Institute of Industrial Science, University of Tokyo, Tokyo, Japan
uday.rage@gmail.com

Abstract. Periodic-frequent patterns are sets of items (values) that periodically appear in a sequence of transactions. The periodicity of a pattern is measured by counting the number of times that its periods (the interval between two successive occurrences of the patterns) are greater than a user-defined $maxPer$ threshold. However, an important limitation of this model is that it can find many patterns having a periodicity that vary widely due to the strict $maxPer$ constraint. But finding stable patterns is desirable for many applications as they are more predictable than unstable patterns. This paper addresses this limitation by proposing to discover a novel type of periodic-frequent patterns in transactional databases, called Stable Periodic-frequent Pattern (SPP), which are patterns having a stable periodicity, and a pattern-growth algorithm named SPP-growth to discover all SPP. An experimental evaluation on four datasets shows that SPP-growth is efficient and can find insightful patterns that are not found by traditional algorithms.

Keywords: Pattern mining · Periodic pattern · Stable periodicity · Lability

1 Introduction

Frequent itemset mining (FIM) [1–4] is a popular data analysis task. The goal is to identify all patterns that frequently appear in records of a transactional database. A pattern is said to be frequent if its support (occurrence frequency) is no less than a user-defined minimum support threshold. Although discovering frequent patterns is useful, too many frequent patterns are often found, and

F. Wotawa et al. (Eds.): IEA/AIE 2019, LNAI 11606, pp. 1–15, 2019.
https://doi.org/10.1007/978-3-030-22999-3_21

many of them are uninteresting to users. To address this issue, several variations of FIM have been developed to select small sets of patterns that are interesting to users based on various constraints. This includes discovering maximal frequent patterns [5], closed frequent patterns [6], high-utility patterns [7], and periodic frequent patterns [8,9,11,12,14,15]. Frequent patterns that periodically occur in a database are called *periodic-frequent patterns* (PFP). Finding these patterns has many practical applications such as for analyzing the behavior of website users and the performance of recommender systems [10]. Finding periodic patterns can also help to understand the purchase behavior of customers by discovering sets of products that are periodically bought. For instance, one may find that a customer buys bread every week. Such pattern can then be used for marketing [8,12].

Several algorithms have been proposed to discover PFP in a transaction database (a sequence of transactions). Most of them measure the periodicity of a pattern by counting the number of times that its periods (number of events between two consecutive occurrences) are greater than a user-specified $maxPer$ threshold. In *full* PFP mining [8,11], a pattern is called periodic if none of its periods are greater than $maxPer$. A drawback of this model is that it is too strict as a pattern is discarded if it has only one period exceeding $maxPer$. For example, a pattern indicating that a customer buys milk every day would be discarded just because the customer skipped one day. An alternative called *partial* PFP mining [14] was then proposed, which relaxed the $maxPer$ constraint to allow a certain number of periods to exceed $maxPer$. But this model is not strict enough, as a pattern may be considered periodic even if it has some very long periods. For example, buying coffee may be considered as periodic if a customer buys it on many consecutive days even if he then did not buy it for a year. Thus, traditional models for discovering PFP patterns are inadequate because the size of periods for some patterns may vary widely in a real-life database but traditional periodicity measures do not take this into account. For several applications such as market basket analysis, it is desirable to identify stable periodic-frequent patterns, that is patterns that have periods that are more or less stable in terms of size over time. Such patterns can be useful to better forecast product demand and improve inventory management strategies.

In this paper, we propose a solution for discovering stable periodic-frequent patterns using a novel measure of stability. This paper has three main contributions. First, a novel measure named *lability* is proposed to assess the stability of patterns. Second, a pattern-growth algorithm, called Stable Periodic-frequent Pattern-growth (SPP-growth), is proposed to efficiently discover the complete set of stable periodic-frequent itemsets. Third, several experiments were conducted on synthetic and real-life datasets to evaluate the efficiency of SPP-growth, and patterns found using the proposed stability measure. Experimental results show that the proposed approach is efficient for finding stable periodic-frequent patterns and that insightful patterns are discovered, which are not discovered using traditional approaches.

The rest of the paper is organized as follows. Section 2 describes related work. Section 3 describes the proposed model of SPPs. Section 4 introduces our algorithm to find all SPPs in transactional databases. Section 5 reports experimental results. Finally, Sect. 6 draws a conclusion.

2 Related Work

The problem of frequent itemset mining is defined as follows [1,2]. Let I be a set of items or symbols. Each subset $X \subseteq I$ is said to be an itemset. The length of an itemset containing k items is said to be k. Furthermore, all itemsets of a length k are called k-itemsets. A *transactional database* $D = \{T_1, T_2, ..., T_n\}$ is a set of transactions where each transaction T_c is an itemset with a unique Transaction IDentifier (TID) c, and where the TID can also represent the transaction time (or timestamp). The support of an itemset X in a database D is denoted as $sup(X)$ and defined as $|\{T|T \in D \wedge X \subseteq T\}|$. In other words, $sup(X) = |g(X)|$, where $g(X)$ is the set of transactions containing X.

Table 1. A transactional database

TID	Itemset	TID	Itemset
1	a, b, c, e	6	b, c, e
2	a, b, c, d	7	b, c, d, e
3	a, b, e	8	a, c
4	c, e	9	a, b, d
5	b, d, e	10	b

For example, consider the database of Table 1, which will be used as running example. This database contains ten transactions $(T_1, T_2, ..., T_{10})$. Transaction T_2 has TID 2 and is a 4-itemset $\{a, b, c, d\}$. The set of transactions containing the itemset $\{a, b\}$ is $g(\{a, b\}) = \{T_1, T_2, T_3, T_9\}$. Hence, the support of $\{a, b\}$ is $sup(\{a, b\}) = |g(\{a, b\})| = 4$.

Definition 1 (Frequent itemset mining). *The problem of frequent itemset mining consists of discovering the frequent itemsets. An itemset X is frequent if $sup(X) \geq minSup$, where $minSup$ is a user-specified minimum support threshold.*

For instance, if $minSup = 5$, there are five frequent itemsets: {a} : 5, {b} : 8, {c} : 6, {e} : 6 and {b,e} : 5, where each itemset X is annotated with $sup(x)$.

Various algorithms have been proposed to discover frequent itemsets, such as Apriori [1] and FP-Growth [2]. However, these algorithms cannot be applied to mine PFPs as they do not evaluate the periodic behavior of patterns. Inspired by studies on frequent itemset mining, researchers have designed algorithms to discover periodic-frequent patterns in transaction databases [2,11–14]. Several applications of mining periodic-frequent patterns were presented in previous

studies [8,10]. The traditional (*full*) periodic-frequent pattern mining model is defined as follows [8].

Definition 2 (Periods of an itemset). *Let there be an itemset X and a database $D = \{T_1,T_2,...,T_n\}$. The set of transactions containing X is denoted as $g(X) = \{T_{gX(1)}, T_{gX(2)}, ..., T_{gX(sup(X))}\}$. Let $T_{gX(i+1)}$ and $T_{gX(i)}$, $i \in [0, sup(X)]$ be two consecutive TIDs where X appears. For an integer $i \geq 0$, the number of transactions or the time difference between $T_{gX(i+1)}$ and $T_{gX(i)}$ is said to be a period of X, defined as $per(X, i) = gX(i+1) - gX(i)$. For simplicity of computation, it is considered that X appears in two additional transactions where $gX(0) = 0$ and $gX(sup(X)+1) = |D|$. The periods of an itemset X is a list of periods defined as $per(X) = \{gX(1) - gX(0), gX(2) - gX(1), ..., gX(sup(X)+1) - gX(sup(X))\}$. Thus, $per(X) = \bigcup_{0 \leq z \leq sup(X)}(gX(z+1) - gX(z))$ and $|per(X)| = |g(X)| + 1 = sup(X) + 1$.*

For example, consider the itemset $\{b, e\}$. This itemset appears in transactions T_1, T_3, T_5, T_6 and T_7, and thus $g(\{b, e\}) = \{T_1, T_3, T_5, T_6, T_7\}$. The periods of this itemset are $per(\{b, e\}) = \{1,2,2,1,1,3\}$.

Definition 3 (Periodic-frequent pattern). *Let $per(X)$ be the set of all periods of X. Then, the periodicity of X can be defined as $maxper(X) = max(per(X))$. An itemset X is a periodic-frequent pattern if $sup(X) \geq minSup$ and $maxper(X) \leq maxPer$, where $minSup$ and $maxPer$ are user-defined thresholds.*

For example, if $minSup = 5$ and $maxPer = 2$, the complete set of ($full$) PFPs is {b}: (8, 2) and {c}: (6, 2), where each PFP X is annotated with a pair ($sup(X)$, $maxper(X)$).

Tanbeer et al. [8] proposed the problem of mining PFPs and the PF-growth algorithm. Then, the MTKPP [11] algorithm was designed, which relies on a depth-first search and a vertical database representation. But these two algorithms have the drawback that a pattern is discarded if only one of its periods exceeds $maxPer$ (i.e., the item {e} has a periodic behavior, but it is regarded as non periodic since it has a period of $3 > maxPer = 2$). Several variations of the above definition were proposed to address some of its limitations. Surana et al. [13] proposed to associate a minimum support threshold and a maximum periodicity to each item, to evaluate each item in a different way. But the number of parameters becomes equal to the number of items. Kiran et al. [14] relaxed the maximum periodicity constraint by considering that a pattern X is (*partial*) periodic if its *periodic-frequency* ($\frac{|\{i|per(X,i) \leq maxPer\}|}{|per(X)|}$) is no less than a user-defined threshold. However, a major drawback of that definition is that a pattern that has some very long periods will still be considered as periodic (i.e., the item {a} is periodic even if it is periodic only in a very short time-interval). In summary, these studies measure the periodicity of a pattern by counting the number of times that its periods are less than $maxPer$ but ignore by how much these periods deviates from $maxPer$. To our best knowledge, this study first considers the problem of finding stable periodic-frequent patterns by taking into account by how much the periods of each pattern deviate from $maxPer$.

3 The Proposed Model

The proposed model defines the concept of stable periodic-frequent patterns. In this model, the definition of periodic-frequent patterns is extended to capture the frequent patterns having a stable periodic behavior. The basic idea of the proposed model is to assess the periodic stability of a pattern by calculating the cumulative sum of the difference between each period length and $maxPer$. The proposed model is defined as follows.

Definition 4 (Lability of an itemset). *Let $T_{gX(i+1)}$ and $T_{gX(i)}$, $i \in [0, sup(X)]$ be two consecutive TIDs where X appears. The i-th lability of X is defined as $la(X,i) = max(0, la(X,i-1) + per(X,i) - maxPer)$, where $la(X,-1) = 0$. Moreover, it can be concisely written as*

$$la(X,i) = max(0, la(X,i-1) + gX(i+1) - gX(i) - maxPer)$$

The lability of an itemset X is a list of periods defined as $la(X) = \{la(X,0), la(X,1), ..., la(X,sup(X))\}$, and $|la(X)| = |per(X)| = sup(X) + 1$.

For example, consider the item {d}. The terms for computing its lability are $la(\{d\},0) = max(0, la(\{d\},-1) + per(\{d\},0) - maxPer) = max(0, 0+1-2) = 0$, $la(\{d\},1) = 1$, $la(\{d\},2) = 1$, $la(\{d\},3) = 1$ and $la(\{d\},4) = 0$. Hence, the lability of $\{d\}$ is $la(\{d\}) = \{0,1,1,1,0\}$.

Based on Definition 4, it can be observed that the periodic behavior of a pattern is stable (lability is zero) if its periods are always no greater than $maxPer$. If a pattern has periods larger than $maxPer$, its lability will increase, and these exceeding values will be accumulated by the lability measure. And if there exists periods of a pattern that are smaller than $maxPer$, its lability will decrease for these periods until it reaches a minimum of zero. Thus, the *lability* of a pattern changes over time depending on its periodic behavior, and each value exceeding $maxPer$ is accumulated. A low lability value (close to zero) means a stable periodic behavior while a high value means an unstable one. Hence, this measure can be used to find stable patterns by limiting the maximum lability.

Definition 5 (Stable periodic-frequent pattern). *Let $la(X)$ be the set of all i-th lability of a pattern X. The stability of X is defined as $maxla(X) = max(la(X))$. An itemset X is a stable periodic-frequent pattern (SPP) if $sup(X) \geq minSup$ and $maxla(X) \leq maxLa$, where $minSup$ and $maxLa$ are thresholds.*

For example, by continuing the previous example, if the user specifies that $maxLa = 1$, the complete set of SPPs is {b}: (8, 0), {c}: (6, 0), {e}: (6, 1) and {b,e}: (5, 1), where each SPP X is annotated with ($sup(X)$, $maxla(X)$).

It is interesting to note that if $maxLa = 0$, SPPs are the traditional PFPs. Thus the proposed SPPs is a generalization of the traditional definition of PFPs.

Definition 6 (Problem definition). *Given a transaction database (D), set of items (I), user-defined minimum support threshold ($minSup$), user-defined maximum periodicity threshold ($maxPer$) and maximum lability threshold ($maxLa$),*

the problem of finding stable periodic-frequent patterns is to discover each pattern X *in* D *such that* $sup(X) \geq minSup$ *and* $maxla(X) \leq maxLa$.

To develop an efficient algorithm for mining SPPs, it is important to design efficient pruning strategies. To use the stability measure for pruning the search space, the following theorem is proposed.

Lemma 1 (Monotonicity of the maximum lability). *Let* X *and* Y *be itemsets such that* $X \subset Y$. *It follows that* $maxla(Y) \geq maxla(X)$.

Proof. Since $X \subset Y$, $g(Y) \subseteq g(X)$. If $g(Y) = g(X)$, then X and Y have the same periods, thus $la(Y) = la(X)$ and $maxla(Y) = maxla(X)$. If $g(Y) \subset g(X)$, then for each transaction $\{T_z | T_z \in g(X) \wedge T_z \notin g(Y)\}$, the corresponding period $per(X, z)$ will be replaced by a larger period $per(Y, z)$. Thus, any period in $per(Y)$ cannot be smaller than a period in $per(X)$. Hence, $maxla(Y) \geq maxla(X)$.

Theorem 1 (Maximum lability pruning). *Let* X *be an itemset appearing in a database* D. X *and its supersets are not SPPs if* $maxla(X) > maxLa$. *Thus, if this condition is met, the search space consisting of* X *and all its supersets can be discarded.*

Proof. By definition, if $maxla(X) > maxLa$, X is not a SPP. By Lemma 1, supersets of X are also not SPPs.

4 The SPP-Growth Algorithm

This subsection introduces the proposed *SPP-growth* algorithm. It performs two steps: (*i*) compressing the database into a stable periodic-frequent tree (SPP-tree) and (*ii*) mining the SPP-tree to find all stable periodic-frequent patterns.

4.1 The SPP-Tree Structure

The SPP-tree structure consists of a prefix-tree and a SPP-list. The SPP-list consists of entries having three fields: item name (i), support (S) and maximum lability (ML). The prefix-tree structure of the SPP-tree is similar to that of the FP-tree [2]. However, to calculate both the support and maximum lability of patterns, the SPP-tree nodes explicitly maintain occurrence information for each transaction by maintaining an occurrence TID (or timestamp) list, called *TID-list* at the last node of every transaction. Hence, two types of nodes are maintained in a SPP-tree: **ordinary** nodes and **tail** nodes. Ordinary nodes are similar to FP-tree nodes [2], whereas tail nodes are the last items of any sorted transaction. The structure of a tail node is $i[t_a, t_b, \cdots, t_c]$, where i is the node's item name and t_j ($j \in [1, n]$) is a TID where item i is the last item. To facilitate tree traversal, each node in the prefix-tree maintains parent, children and node traversal pointers. Besides, unlike FP-tree nodes, SPP-tree nodes do not maintain a support count value. To increase the likelihood of obtaining a compact tree, items in the prefix-tree are arranged in descending order of their **support** of SPP. The next paragraphs explain how a SPP-tree is constructed and mined to extract SPPs.

i	S	ML	t_{cur}
a	1	0	1
b	1	0	1
c	1	0	1
e	1	0	1

(a)

i	S	ML	t_{cur}
a	2	0	2
b	2	0	2
c	2	0	2
e	1	0	1
d	1	0	2

(b)

i	S	ML	t_{cur}
a	5	3	9
b	8	0	10
c	6	0	8
e	6	0	7
d	4	1	9

(c)

i	S	ML
a	5	3
b	8	0
c	6	0
e	6	1
d	4	1

(d)

i	S	ML
b	8	0
c	6	0
e	6	1

(e)

Fig. 1. The SPP-list of of Table 1 after scanning (a) the first transaction, (b) the second transaction, (c) the entire database, (d) after adding $t = gX(sup(X) + 1)$ to each item, and (e) the final SPP-list containing the sorted list of items.

4.2 Constructing an SPP-tree

To construct a SPP-list, a temporary array t is used to record the current TID (or timestamp) of an item. Algorithm 1 describes the steps for constructing a SPP-list. Consider the database of Table 1 and that *minSup*, *maxPer* and *maxLa* values are set to 5, 2 and 1, respectively. Figures 1(a)–(c) show the construction of the SPP-list after scanning the first, second, and all transactions of the database, respectively (line 2 to 9 of Algorithm 1). Figures 1(d) shows the result of adding $t = g(sup(X) + 1)$ to every item (line 10 of Algorithm 1). Figures 1(e) shows the SPP-list containing stable periodic-frequent items, sorted by descending order of *support* (line 10 of Algorithm 1).

After finding stable periodic-frequent items, the database is scanned again to construct the prefix-tree of the SPP-tree (Algorithm 2). The construction of the prefix-tree is similar to how a FP-tree is constructed [2]. But it has to be noted that only **tail** nodes of a SPP-tree maintain TIDs (or timestamps). Figure 2(a)–(e) show the construction of the SPP-tree after scanning the first, second, eighth, and all transactions of the database. In a SPP-tree, an item header table is built so that each item points to its occurrences in the tree via a chain of node-links to facilitate tree traversal. For simplicity, we do not show these node-links in the illustrations. They are created in the same way as for the FP-tree.

4.3 Mining an SPP-tree

The SPP-tree is then mined as follows. A bottom-up scan is done to browse each stable periodic-frequent item of the header table of the SPP-tree. The conditional pattern base of each item is constructed (a projected database, consisting of the set of prefix paths of the SPP-tree co-occurring with the suffix itemset) to collect the TIDs of its ancestors and calculate its *SPP-list*. Then, the item's conditional SPP-tree is constructed (where ancestors that have a support less than *minSup* or a *maxla* larger than *maxSup* are pruned), and mining is pursued recursively on the resulting tree. The algorithm is said to be of type pattern-growth because it recursively grows each SPP by appending single items from

Algorithm 1. Construction of an SPP-list

input : D: a transactional database,
$minSup$, $maxPer$, $maxLa$: the user-specified thresholds
output: the SPP-list containing the sorted list of items

1 Record the TID $t_{last}(i)$ of the last transaction containing each item i in a temporary array. Set the maximum lability $ML(i)$, the lability $la(i)$ and support $S(i)$ of each item i to 0;
2 **foreach** *transaction* $T \in D$ *with TID* t_{cur} **do**
3 **foreach** *item* $i \in T$ **do**
4 $S(i) = S(i) + 1$;
5 $la(i) = max(0, la(i) + t_{cur} - t_{last}(i) - maxPer)$;
6 $ML(i) = max(ML(i), la(i))$;
7 $t_{last}(i) = t_{cur}$;
8 **end**
9 **end**
10 Set $t_{cur} = |D|$ and update each item in the SPP-list. Remove each item i such that $S(i) < minSup$ or $ML(i) > maxLa$ from the SPP-list. Consider the remaining items of the SPP-list as stable periodic-frequent items and sort them by descending order of support.

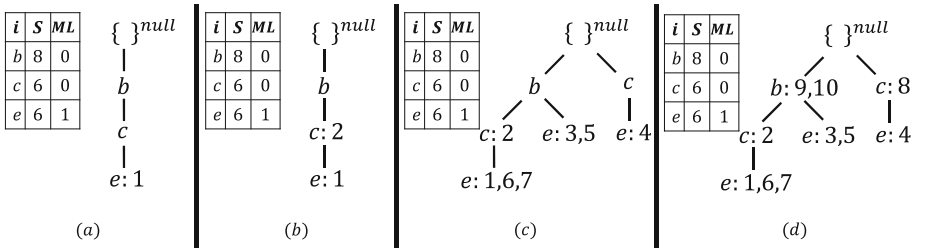

Fig. 2. Construction of an SPP-tree after scanning the (a) first transaction, (b) second transaction, (c) eighth transaction, and (d) the entire database.

its generated conditional SPP-tree. The procedure to discover SPPs from the SPP-tree is shown in Algorithm 3.

To illustrate how the SPP-tree is processed during mining, Fig. 3(a) shows the SPP-tree of Fig. 2(d) after removing the bottommost item e. Note that the *TID*-list of e has been pushed-up to its parent node since the node e was a *tail*-node in the original SPP-tree. Besides, Fig. 3(b) shows the prefix-tree of suffix item e. A *conditional pattern base* is a path from the root node to a leaf node. The conditional pattern bases of e are thus {b:3,5}, {bc:1,6,7} and {c:4}. Using the prefix-tree of e, it is rather simple to compute the *maxLa* and *support* of each item that is co-occurring with the suffix itemset. The pruning conditions are checked and nodes not respecting the thresholds are deleted from the SPP-tree, while stable periodic patterns are output. The resulting tree is shown in Fig. 3(c). The item c was deleted because its support is less than 5,

Algorithm 2. Construction of an SPP-tree

input : D: a transactional database,
SPP-list: contains stable periodic-frequent items, their S and ML

Create the root of the SPP-tree, T, and label it with "null" ;
foreach *transaction* $T \in D$ *with TID* t_{cur} **do**
    Sort stable periodic-frequent items in T according to the order of *SPP-list*. Let the sorted candidate item list be $[p|P]$, where p is the first item and P is the remaining list. Call $insert\_tree([p|P], ts_{cur}, T)$, which is performed as follows. If T has a child N such that $N.item\text{-}name \neq p.item\text{-}name$, then create a new node N. Link its parent to T. Let its node-link be linked to nodes with the same *item-name* via the node-link structure. Remove p from $[p|P]$. If P is empty, add t_{cur} to the leaf node; else, call $insert\_tree(P, ts_{cur}, N)$ recursively.
end

Algorithm 3. The SPP-Growth algorithm

input : T: a SPP-tree, α: suffix itemset (initial value is $\emptyset$)
output: the set of SPPs

while *T's header table* $T^{ht} \neq \emptyset$ **do**
    $i = T^{ht}[T^{ht}.size - 1]$;
    $\beta = \alpha \cup i$, output β;
    Traverse the node-link of i to construct β's conditional pattern base and collect its TIDs where β has appeared in D and calculate $SPP\text{-}list_{\beta}$;
    if $SPP\text{-}list_{\beta} \neq \emptyset$ **then**
        Construct β's conditional tree T_{β} and call $SPP\text{-}Growth(T_{\beta}, \beta)$;
    end
    Remove i from T and push i's TIDs to its parent nodes.
end

while *be* is output as a stable periodic pattern. Then same process of creating a prefix-tree and its corresponding conditional tree is repeated in the same way to consider other pattern extensions. The whole process of mining patterns using each item is repeated if the header table of the SPP-tree is empty. Because the SPP-growth algorithm starts from SPP-tree of single items and recursively explores the conditional SPP-tree of patterns and only prunes the search space using Theorem 1, it can be seen that this procedure is correct and complete to discover all SPPs.

5 Experimental Evaluation

Since SPP-growth is the first algorithm for mining SPPs, its performance is not compared with other algorithms. SPP-growth is implemented in Java. Experiments were performed on a computer having a 64 bit Xeon E3-1270 3.6 GHz

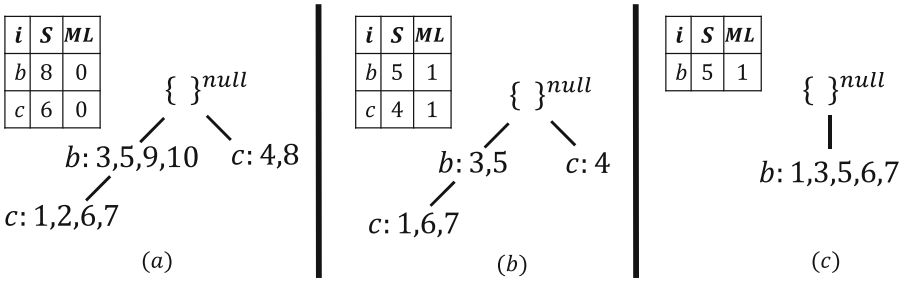

Fig. 3. Mining stable periodic-frequent patterns using suffix item e. (a) The SPP-tree after removing the item e, (b) prefix-tree of suffix item e, (c) conditional tree of suffix item e.

CPU, running Windows 10 and having 64 GB of RAM. The algorithm is evaluated in terms of performance on both synthetic (*T10I4D100K*) and real-world (*mushroom*, *OnlineRetail* and *kosarak*) datasets, obtained from the SPMF website [16]. Characteristics of the datasets are presented in Table 2, where $|D|$, $|I|$, T_{min}, T_{max} and T_{avg} denote the number of transactions, distinct items, minimum transaction length, maximum transaction length and average transaction length, respectively. Datasets were chosen because they have different characteristics (dense/sparse datasets, long/short transactions, few/many items).

Table 2. Characteristics of the datasets

Dataset	$\|D\|$	$\|I\|$	T_{min}	T_{max}	T_{avg}
T10I4D100K	100,000	870	1	29	10
mushroom	8,124	119	23	23	23
kosarak	990,002	41,270	1	2,498	8
OnlineRetail	541,909	2,603	1	1,108	1

Figure 4 shows the runtime requirements of the SPP-growth algorithm for different *minSup*, *maxPer* and *maxLa* values on *mushroom* and $T10I4D100K$, respectively. In these charts, values for the maximum lability thresholds ($maxLa$) are shown on the x axis, while the y axis denotes execution times. The notation S-P denotes the SPP-growth algorithm with $minSup = S$ and $maxPer = P$. The following two observations are drawn from Fig. 4:

- Increasing $maxLa$ often increase the runtime. The reason is that increasing $maxLa$ increases the range of lability values accepted for SPPs. Thus, more patterns must be considered in the search space.
- The SPP-growth algorithm has better performance on the sparse dataset than on the dense dataset. The reason is that patterns in sparse datasets are

more likely to be unstable. Hence, SPP-growth algorithm can eliminate many candidate patterns on such datasets.

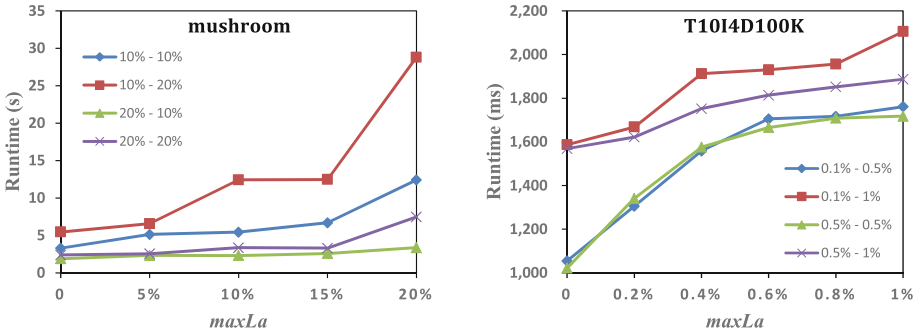

Fig. 4. Execution times for different parameter values

Figure 5 shows the number of SPPs generated for different *minSup*, *maxPer* and *maxLa* values for *mushroom* and $T10I4D100K$, respectively. The following observations can be drawn:

- For fixed *maxLa* and *maxPer* values, increasing *minSup* may decrease the number of stable periodic-frequent patterns. The reason is that some patterns will then fail to satisfy the higher *minSup* threshold value.
- Similarly, for fixed *maxLa* and *minSup* values, increasing *maxPer* may increase the number of SPPs. The reason is that as *maxPer* is increased, frequent patterns having longer periods may become stable periodic-frequent patterns.
- For the $T10I4D100K$ dataset, the pattern count increases rapidly as the *support* and *maxPer* threshold are increased at the same time. This is

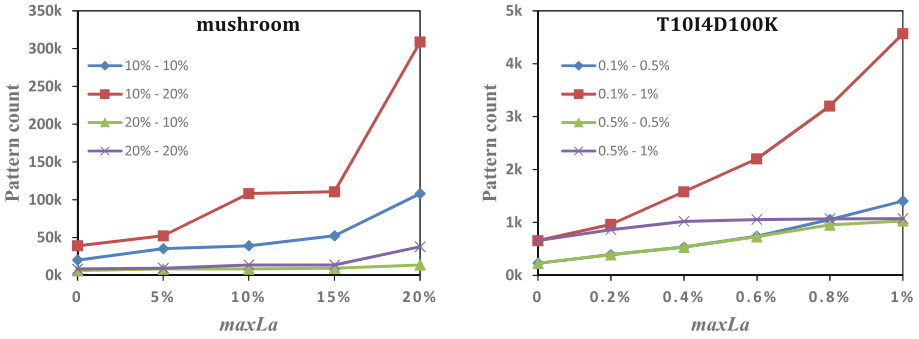

Fig. 5. Number of SPPs found for different parameter values

because patterns in sparse datasets are more likely to be candidate patterns. Hence, increasing the *support* and *maxPer* thresholds at same time results in more candidate patterns that need to be considered as potential SPPs.

We also evaluated the proposed algorithm's scalability in terms of execution time, number of SPPs found and number of tree nodes when the number of transactions is varied. For this experiment, the real *kosarak* dataset is used, since it has a large number of distinct items and transactions. The dataset was divided into five parts and then the performance of the algorithm was measured after adding each part to the previous ones. Figure 6 shows the experiment's results for $minSup = 0.1\%$, $maxPer = 2\%$ and $maxLa = 1\%$. It is clear that the runtime, number of patterns and number of nodes increase along with the database size. This is reasonable because the $maxLa$ and $maxPer$ of patterns also increase with the database size. Hence, the algorithm finds more patterns and spend time to build additional tree nodes when size is increased.

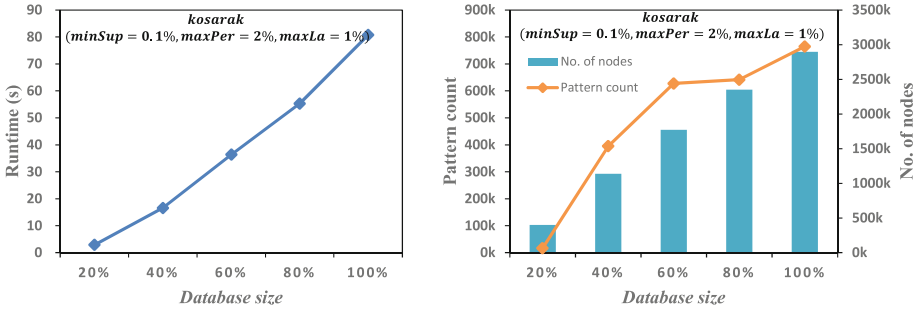

Fig. 6. Scalability of SPP-growth when varying the database size

Table 3. Comparison of peak memory usage.

Algorithms			*T10I4D100K* (MB)	No. of nodes
minSup	*maxPer*	*maxLa*		
0.1%	0.5%	0%	263	334,153
0.1%	0.5%	0.4%	296	575,854
0.1%	1%	0%	324	596,766
0.1%	1%	0.4%	421	652,840
0.2%	0.5%	0%	263	334,153
0.2%	0.5%	0.4%	296	575,854
0.2%	1%	0%	323	596,351
0.2%	1%	0.4%	389	638,054

In another experiment, the peak memory usage of SPP-growth was recorded for different parameter values on $T10I4D100K$. Results (Table 3) show that SPP-growth consumes less memory for high $minSup$, low $maxPer$ and low $maxLa$ values. This is reasonable as more patterns can be pruned.

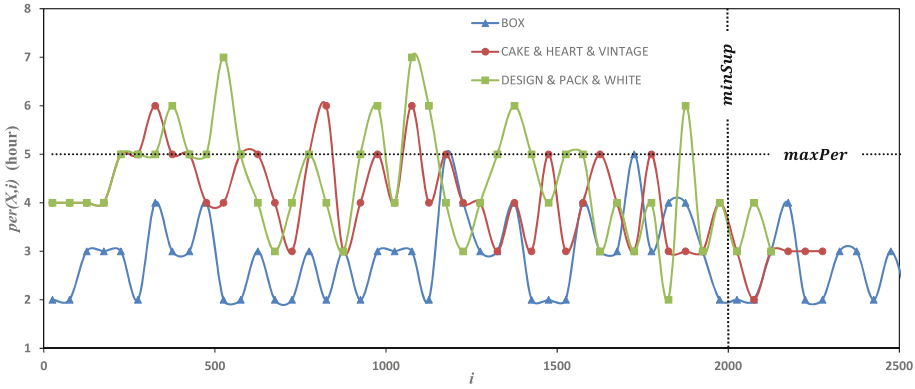

Fig. 7. Periods of some interesting SPPs found in *OnlineRetail*

We also analyzed the SPPs found in the real *OnlineRetail* sale dataset to assess their usefulness. *OnlineRetail* contains transactions of a UK-based online store from 01/12/2010 to 09/12/2011. Data was segmented into hours to obtain 2,975 non empty transactions. For $minSup = 2,000$, $maxPer = 5$ h and $maxLa = 2$ h, 32,284 SPPs are found, and only 197 PFPs for $maxLa = 0$ h. This shows that many stable patterns can be saved using the proposed algorithm. Figure 7 shows some found SPPs, which are {Box}: (2553, 0h), {Cake, Heart, Vintage}: (2284, 1 h) and {Design, Pack, White}: (2131, 2 h), where each SPP X is annotated with ($sup(X)$, $maxla(X)$). The X-axis indicates period numbers of patterns, and the Y-axis indicates the value $per(X, i)$ for the i-th period. Note that to reduce the number of points on that chart, only the maximum value for each group of 50 periods is shown in Fig. 7. It can be observed that frequent patterns exceeding $maxPer$ having a stable periodic behavior are obtained, while such patterns would be ignored by traditional PFP mining algorithms due to the $maxPer$ constraint. The stable patterns found about the sale of products are also deemed interesting as they indicate stable sale trends. Such information could be used to forecast product sales.

6 Conclusion

This paper proposed a novel problem of mining stable periodic-frequent patterns. A new $maxLa$ measure has been designed to assess the stability of an itemset's periodic behavior in a database. A pattern-growth algorithm has also been proposed to find SPPs. An experimental evaluation on both synthetic and

real datasets shows that SPP-growth is efficient and can find useful patterns. For future work, we plan to adapt the concepts of stability to mine other types of patterns such as stable periodic sequential patterns.

References

1. Bodon, F., Schmidt-Thieme, L.: The relation of closed itemset mining, complete pruning strategies and item ordering in apriori-based FIM algorithms. In: Jorge, A.M., Torgo, L., Brazdil, P., Camacho, R., Gama, J. (eds.) PKDD 2005. LNCS (LNAI), vol. 3721, pp. 437–444. Springer, Heidelberg (2005). https://doi.org/10.1007/11564126_43
2. Han, J., Pei, J., Yin, Y.: Mining frequent patterns without candidate generation. In: Proceedings of 26th ACM SIGMOD International Conference on Management of Data, pp. 1–12 (2000)
3. Zaki, M.J., Gouda, K.: Fast vertical mining using diffsets. In: Proceedings of 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 326–335 (2003)
4. Fournier-Viger, P., Lin, J.C.-W., Vo, B., Truong, T.C., Zhang, J., Le, H.B.: A survey of itemset mining. Wiley Interdisc. Rev. Data Min. Knowl. Discov. **7**(4), e1207 (2017)
5. Gouda, K., Zaki, M.J.: Efficiently mining maximal frequent itemsets. In: Proceedings of 17th IEEE International Conference on Data Mining, pp. 163–170 (2001)
6. Pasquier, N., Bastide, Y., Taouil, R., Lakhal, L.: Discovering frequent closed itemsets for association rules. In: Beeri, C., Buneman, P. (eds.) ICDT 1999. LNCS, vol. 1540, pp. 398–416. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-49257-7_25
7. Fournier-Viger, P., Wu, C.-W., Zida, S., Tseng, V.S.: FHM: faster high-utility itemset mining using estimated utility co-occurrence pruning. In: Andreasen, T., Christiansen, H., Cubero, J.-C., Raś, Z.W. (eds.) ISMIS 2014. LNCS (LNAI), vol. 8502, pp. 83–92. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08326-1_9
8. Tanbeer, S.K., Ahmed, C.F., Jeong, B.S., Lee, Y.K.: Discovering periodic-frequent patterns in transactional databases. In: 13th Proceedings of Pacific-Asia Conference on Knowledge Discovery and Data Mining, pp. 242–253 (2009)
9. Kiran, R.U., Kitsuregawa, M., Reddy, P.K.: Efficient discovery of periodic-frequent patterns in very large databases. J. Syst. Softw. **112**, 110–121 (2016)
10. Fong, A.C.M., Zhou, B., Hui, S.C., Hong, G.Y., Do, T.: Web content recommender system based on consumer behavior modeling. IEEE Trans. Consum. Electron. **57**(2), 962–969 (2011)
11. Amphawan, K., Lenca, P., Surarerks, A.: Mining top-K periodic-frequent pattern from transactional databases without support threshold. In: Papasratorn, B., Chutimaskul, W., Porkaew, K., Vanijja, V. (eds.) IAIT 2009. CCIS, vol. 55, pp. 18–29. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-10392-6_3
12. Fournier-Viger, P., Lin, J.C.-W., Duong, Q.-H., Dam, T.-L.: PHM: mining periodic high-utility itemsets. In: Perner, P. (ed.) ICDM 2016. LNCS (LNAI), vol. 9728, pp. 64–79. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-41561-1_6
13. Surana, A., Kiran, R.U., Reddy, P.K.: An efficient approach to mine periodic-frequent patterns in transactional databases. In: Cao, L., Huang, J.Z., Bailey, J., Koh, Y.S., Luo, J. (eds.) PAKDD 2011. LNCS (LNAI), vol. 7104, pp. 254–266. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-28320-8_22

14. Kiran, R.U., Venkatesh, J.N., Fournier-Viger, P., Toyoda, M., Reddy, P.K., Kitsuregawa, M.: Discovering periodic patterns in non-uniform temporal databases. In: Kim, J., Shim, K., Cao, L., Lee, J.-G., Lin, X., Moon, Y.-S. (eds.) PAKDD 2017. LNCS (LNAI), vol. 10235, pp. 604–617. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-57529-2_47
15. Fournier-Viger, P., Li, Z., Lin, J.C.-W., Kiran, R.U., Fujita, H.: Discovering periodic patterns common to multiple sequences. In: Ordonez, C., Bellatreche, L. (eds.) DaWaK 2018. LNCS, vol. 11031, pp. 231–246. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-98539-8_18
16. Fournier-Viger, P., Gomariz, A., Gueniche, T., Soltani, A., Wu, C., Tseng, V.S.: SPMF: a Java open-source pattern mining library. J. Mach. Learn. Res. (JMLR) **15**, 3389–3393 (2014)