Information Sciences 489 (2019) 205–226

Contents lists available at ScienceDirect

Information Sciences

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

Efficient algorithms to identify periodic patterns in multiple sequences

Philippe Fournier-Viger[a,*], Zhitian Li[b], Jerry Chun-Wei Lin[c], Rage Uday Kiran[d], Hamido Fujita[e]

[a] School of Humanities and Social Sciences, Harbin Institute of Technology (Shenzhen), Shenzhen, Guangdong 518055, China
[b] School of Computer Sciences and Technology, Harbin Institute of Technology (Shenzhen), Shenzhen, Guangdong 518055, China
[c] Department of Computing, Mathematics and Physics, Western Norway University of Applied Sciences (HVL), Bergen 5020, Norway
[d] Institute of Industrial Science, University of Tokyo, Tokyo 153-8505, Japan
[e] Iwate Prefectural University, Morioka 020-8550, Japan

A R T I C L E   I N F O

Article history:
Received 13 June 2018
Revised 21 March 2019
Accepted 22 March 2019
Available online 23 March 2019

Keywords:
Periodic pattern
Frequent pattern
Sequence periodic ratio
Sequences

A B S T R A C T

Periodic pattern mining is a popular data mining task, which consists of identifying patterns that periodically appear in data. Traditional periodic pattern mining algorithms are designed to find patterns in a single sequence. However, in several domains, it is desirable to discover patterns that are periodic in many sequences. An example of such application is market basket analysis. Given a database of sequences of transactions made by customers, discovering sets of items that are periodically bought by customers can help understand customer behavior. To discover periodic patterns common to multiple sequences, this paper extends the traditional problem of mining periodic patterns in a sequence. Two novel measures are defined called the standard deviation of periods and the sequence periodic ratio. Two algorithms are proposed to mine these patterns efficiently called MPFPS$_{BFS}$ and MPFPS$_{DFS}$, which perform a breadth-first search and depth-first search, respectively. Because the sequence periodic ratio is neither monotone nor anti-monotone, these algorithms rely on a novel upper-bound called *boundRa* and two novel search space pruning properties to find periodic patterns efficiently. The algorithms have been evaluated on multiple datasets. Results show that they are efficient and can filter numerous non periodic itemsets to identify periodic patterns.

1. Introduction

Frequent Pattern Mining (FPM) is a popular data mining problem, which consists of finding frequently appearing patterns in a database [6,19,20]. FPM has been initially proposed for analyzing customer transaction databases to identify frequent purchases made by customers. Nowadays, FPM is used in many other fields for various tasks such as community discovery [6], image classification [11], malware detection [9], e-learning [37] and activity monitoring [34]. Many algorithms have been developed to efficiently mine frequent patterns and various measures have been proposed to identify interesting patterns [20]. A limitation of traditional FPM algorithms is that they ignore the sequential ordering between transactions or

[*] Corresponding author.
E-mail addresses: philfv@hit.edu.cn (P. Fournier-Viger), tymonlee1@163.com (Z. Li), jerrylin@ieee.org (J.C.-W. Lin), uday_rage@tkl.iis.u-tokyo.ac.jp (R.U. Kiran), hfujita-799@acm.org (H. Fujita).

https://doi.org/10.1016/j.ins.2019.03.050
0020-0255/© 2019 Elsevier Inc. All rights reserved.

Table 1
A customer sequence database.

ID	Sequence
0	⟨(*bread, milk*), (*apple*), (*bread, apple, milk*), (*kiwi, peach*), (*bread, milk, egg*)⟩
1	⟨(*apple, milk*), (*bread, apple, milk*), (*wine, egg*), (*bread, milk, pen*), (*kiwi*), (*bread, milk, egg*)⟩
2	⟨(*bread, milk*), (*pen, book*), (*wine, bread, milk*), (*kiwi, peach*), (*bread, milk, peach*)(*pen, book*)⟩
3	⟨(*egg, wine*), (*kiwi, pen, book*), (*wine, bread*), (*kiwi, egg, peach*)⟩
4	⟨(*bread, apple, milk*), (*apple, wine*), (*bread, milk*), (*egg*), (*bread, kiwi, milk*), (*pen, book*)⟩
5	⟨(*milk, egg, apple*), (*pen, book*), (*wine, milk*), (*apple, peach*)⟩
6	⟨(*apple, peach*), (*pen, book, apple*), (*egg, pen, book*), (*kiwi, apple*)⟩
7	⟨(*apple, pen, book, milk*), (*peach, kiwi*), (*egg, milk, bread*), (*kiwi, apple, milk*)⟩

events in a database. Thus, these algorithms can be used to find frequent associations between items or events but provide no information about their order. For example, frequent itemset mining algorithms can discover that several customers buy some sets of items frequently but information about the order of these purchases is ignored. In many domains such as bioinformatics [45], e-learning [22,48], text-analysis [39] and energy reduction in smarthomes [40], considering the sequential ordering is important and can be used to reveal more meaningful patterns. For instance, in market basket analysis, it can be found that many customers have the same sequential behavior over time. Identifying patterns having information about the sequential order can be used to improve sales and marketing strategies.

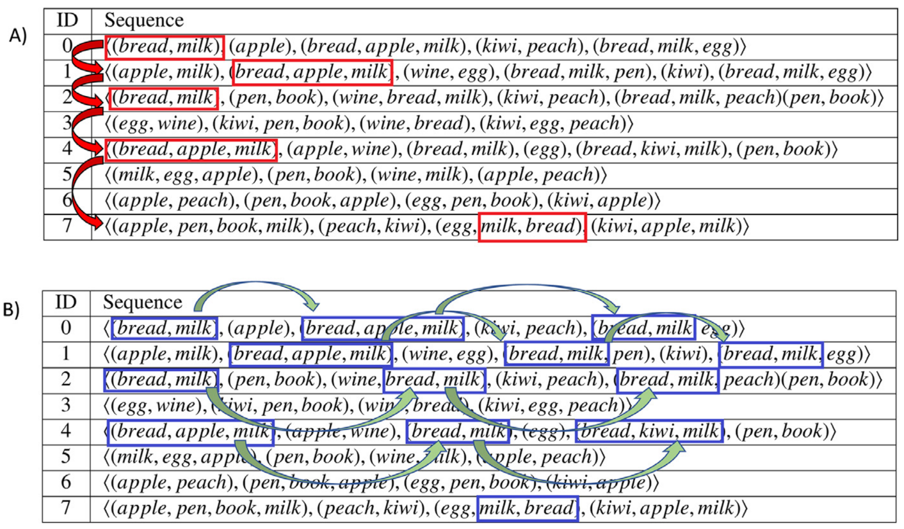
In recent years, numerous studies have proposed algorithms to find patterns that take the order of events or transactions into account. One of the most popular task is Sequential Pattern Mining (SPM), which generalizes the problem of frequent itemset mining to find frequent subsequences in a set of sequences of transactions [2,10,22,39,40,45,48]. Even though many SPM algorithms have been developed and various extensions of SPM have been proposed and used in many applications, a major limitation of SPM algorithms is that they cannot be used to discover patterns that recurrently appear in the data. However, recurring patterns are found in many domains [24,42]. For example, in a gene sequence, a set of DNA molecules may carry tremendous information if it appears regularly after several DNA molecules. Detecting sets of repeating DNA molecules is necessary to find what leads to some external expressions. Another example, is to identify products that are regularly bought by some customers (e.g. every week or month) to promote the sale of groups of items.

To find recurring patterns in a sequence, the task of periodic pattern mining was proposed [3,4,17,18,29,30,42,43]. To discover periodic patterns, the user generally needs to set a parameter called the maximum period threshold, and provide a sequence of transactions. Then, a periodic pattern mining algorithm outputs all patterns such that the number of transactions or events between two occurrences of the pattern is never greater than the maximum period threshold. For example, consider that the maximum period threshold is set to two weeks. It could then be found in a sequence of activities made by a user that the user goes to cinema at least once every two weeks.

Although periodic pattern mining has many applications [43], it is designed to find patterns in a single sequence. However, periodic patterns also appear in sets of sequences (a sequence database). For example, it is desirable to discover periodic behavior of not just one customer, but common to several customers. To our best knowledge, only one algorithm, named PHUSPM, was proposed to mine periodic patterns in a sequence database [8]. However, this algorithm simply applies the same periodicity measures as algorithms for discovering patterns in a single sequence. As a result, PHUSPM treats a sequence database as a single sequence to find patterns that regularly appear from the point of view of sequences but not transactions contained in these sequences. Thus, PHUSPM does not consider whether a pattern is periodic in each sequence. But finding patterns that are periodic in several sequences is useful. For example, consider sequences of customer transactions in a retail store, such as the one illustrated in Table 1. This database contains seven sequences, each representing a customer. The first sequence means that a customer bought the items *bread* and *milk* together, then *apple*, then *bread* with *apple* and *milk*, followed by *kiwi* and *peach*, followed by *bread, milk* and *egg*. The PHUSPM algorithm could find that *bread* and *milk* are periodically sold by the store (appear periodically in the database when considering all customers) but would fail to find that many customers periodically buy bread and milk. Finding such information is useful for designing effective sale and marketing strategies to target a group of customers based on their common periodic behavior.

To address the above limitations of previous work, this paper proposes the task of mining Periodic Frequent Patterns common to multiple Sequences (PFPS), which considers the periodicity of patterns in each sequence and their frequency in the overall database. Moreover, an efficient algorithm is presented to mine PFPS. The contributions of this paper are as follows:

- The problem of mining periodic frequent patterns common to multiple sequences is defined, and its properties are studied. Note that an early version of this work was published in a conference paper [16].
- To evaluate the periodicity of patterns in each sequence, a new measure is defined, which is the standard deviation of periods, to find patterns that occur with regularity. Moreover, a novel periodicity measure named *Sequence Periodic Ratio* (SPR) is defined to find patterns that are periodic in multiple sequences. To effectively reduce the search space, an upper-bound on the SPR called *boundRa* is developed and two novel pruning properties are proposed. The proposed measures allows to find patterns that are periodic in many sequences (intra-sequence periodicity). This is different from the PHUSPM algorithm, which evaluate periodicity between sequences (inter-sequence periodicity). This is illustrated



A)

ID	Sequence
0	⟨(*bread*, *milk*), (*apple*), (*bread*, *apple*, *milk*), (*kiwi*, *peach*), (*bread*, *milk*, *egg*)⟩
1	⟨(*apple*, *milk*), (*bread*, *apple*, *milk*), (*wine*, *egg*), (*bread*, *milk*, *pen*), (*kiwi*), (*bread*, *milk*, *egg*)⟩
2	⟨(*bread*, *milk*), (*pen*, *book*), (*wine*, *bread*, *milk*), (*kiwi*, *peach*), (*bread*, *milk*, *peach*)(*pen*, *book*)⟩
3	⟨(*egg*, *wine*), (*kiwi*, *pen*, *book*), (*wine*, *bread*), (*kiwi*, *egg*, *peach*)⟩
4	⟨(*bread*, *apple*, *milk*), (*apple*, *wine*), (*bread*, *milk*), (*egg*), (*bread*, *kiwi*, *milk*), (*pen*, *book*)⟩
5	⟨(*milk*, *egg*, *apple*), (*pen*, *book*), (*wine*, *milk*), (*apple*, *peach*)⟩
6	⟨(*apple*, *peach*), (*pen*, *book*, *apple*), (*egg*, *pen*, *book*), (*kiwi*, *apple*)⟩
7	⟨(*apple*, *pen*, *book*, *milk*), (*peach*, *kiwi*), (*egg*, *milk*, *bread*), (*kiwi*, *apple*, *milk*)⟩

B)

ID	Sequence
0	⟨(*bread*, *milk*), (*apple*), (*bread*, *apple*, *milk*), (*kiwi*, *peach*), (*bread*, *milk*, *egg*)⟩
1	⟨(*apple*, *milk*), (*bread*, *apple*, *milk*), (*wine*, *egg*), (*bread*, *milk*, *pen*), (*kiwi*), (*bread*, *milk*, *egg*)⟩
2	⟨(*bread*, *milk*), (*pen*, *book*), (*wine*, *bread*, *milk*), (*kiwi*, *peach*), (*bread*, *milk*, *peach*)(*pen*, *book*)⟩
3	⟨(*egg*, *wine*), (*kiwi*, *pen*, *book*), (*wine*, *bread*), (*kiwi*, *egg*, *peach*)⟩
4	⟨(*bread*, *apple*, *milk*), (*apple*, *wine*), (*bread*, *milk*), (*egg*), (*bread*, *kiwi*, *milk*), (*pen*, *book*)⟩
5	⟨(*milk*, *egg*, *apple*), (*pen*, *book*), (*wine*, *milk*), (*apple*, *peach*)⟩
6	⟨(*apple*, *peach*), (*pen*, *book*, *apple*), (*egg*, *pen*, *book*), (*kiwi*, *apple*)⟩
7	⟨(*apple*, *pen*, *book*, *milk*), (*peach*, *kiwi*), (*egg*, *milk*, *bread*), (*kiwi*, *apple*, *milk*)⟩

Fig. 1. Illustration of (A) inter-sequence periodicity of {*bread, milk*} as measured by PHUSPM and (B) intra-sequence periodicity as measured by the proposed algorithms.

in Fig. 1 for the itemset {*bread, milk*}, where inter-sequence periodicity is evaluated vertically (Fig. 1 A), while intra-sequence periodicity is evaluated horizontally (Fig. 1 B). Thus, while PHUSPM finds that {*bread, milk*} is regularly sold when considering all customers, the proposed measures allows to find that several customers each regularly buy {*bread, milk*}.

- Two algorithms, respectively named MPFPS$_{BFS}$ and MPFPS$_{DFS}$, are presented to efficiently find all periodic frequent patterns common to multiple sequences, which relies on a novel PFPS-list structure to avoid repeatedly scanning the database.
- An experimental evaluation on several real and synthetic datasets reveals that the proposed algorithms are efficient and can filter many non periodic patterns. Thus, a small set of patterns can be shown to the user.

The rest of this paper is organized as follows. Section 2 reviews related work. Section 3 defines the problem of mining patterns that are periodic in multiple sequences. Section 4 presents the proposed algorithms. Section 5 describes the experimental evaluation. Finally, Section 6 draws the conclusion and discusses future work.

2. Related work

This section provides an overview of related work related to itemset mining, pattern mining in sequences and periodic pattern mining.

2.1. Frequent itemset mining

Discovering patterns in databases is an important subfield of data mining. One of the most important pattern mining problem is Frequent Itemset Mining (FIM). Given a parameter called the minimum support (*minsup* threshold) and a customer transaction database, FIM consists of discovering all sets of itemsets (itemsets) purchased by at least *minsup* customers [1,20]. The occurrence frequency of an itemset is called its support. In the last decades, FIM has been applied in many fields such as e-learning, malware detection and image classification. To discover frequent itemsets, the Apriori algorithm was first designed [1]. This algorithm explores the search space of itemsets using a breadth-first search. It first generates patterns each containing an item, and then combines them to generate patterns containing two items. This process is then repeated recursively to find larger patterns. For each generated pattern, Apriori scans the database to calculate its support and determine if it is a frequent patterns. Although Apriori can discover all frequent itemsets in a database, repeatedly scanning the database results in poor performance for large databases. The Eclat [46] algorithm addresses this issue by creating a vertical structure called *id-list* for each candidate itemset, which can be built from the id-lists of other patterns to avoid scanning the database many times. Eclat performs a depth-first search and divides the search space into equivalence classes [46]. Thereafter, several other FIM algorithms have been proposed. The FP-growth algorithm utilizes a frequent pattern tree (also called FP-tree) to mine patterns, which is a concise and lossless database representation [26]. The LCM [44] algorithm utilizes an horizontal database but reduce the cost of database scans by merging identical transactions and performing database projections.Several other algorithms have been developped in recent years, and several variations of the itemset mining problem have been proposed [20]. However, most studies on FIM do not consider the ordering between transactions.

2.2. Frequent pattern mining in sequences

To consider the sequential ordering between transactions, FIM has been extended as the task of Sequential Pattern Mining (SPM), which consists of discovering subsequences appearing in at least *minsup* sequences, where *minsup* is an integer set by the user [19,41]. The first algorithm for this problem is AprioriAll, which extends Apriori. The mining process of AprioriAll is mostly the same as Apriori excepts that the order of the last two elements of patterns are considered when generating new candidate patterns [2]. An improved version of AprioriAll named GSP was then proposed to handle additional constraints such as time constraints and handling a user-defined taxonomy of items. To reduce the cost of database scans as larger patterns are explored, the FreeSpan algorithm was developped, which applies the concept of database projection [25]. An improved version of FreeSpan named PrefixSpan was then introduced to only project suffixes having the same prefix to obtain projected databases [27]. Similar to the Eclat algorithm for FIM, several sequential pattern mining have adopted a vertical database representation, such as SPAM [5], PRISM [23] and CM-SPADE [13]. Sequential pattern mining is an active research area with numerous applications [19].

Studies have also investigated the discovery of many other types of patterns in sequences. Episode mining and episode rule mining consists of discovering patterns that have a large occurrence count and confidence in a single sequence [36,49]. The concept of episode rule has also been generalized to multiple sequences in the problem of sequential rule mining, which consists of discovering rules indicating that if some events occur, some other events will then occur [15,21,28,35]. Among these studies, some consider finding patterns accross multiple sequences instead of common to multiple sequences [28,35]. Another emerging topic is to discover recent events in a single sequence to find up-to-date information for decision making [33]. Even though, there have been many studies for mining various kinds of patterns in sequences, few are designed to mine periodic patterns.

2.3. Periodic pattern mining

FIM has been extended to discover periodic patterns in a single sequence. A frequent itemset is said to be periodic in a sequence if it appears multiple times and the time between each occurrence is less than a user-defined maximum periodicity threshold. More efficient algorithms have been designed, and variations of the problem of mining periodic patterns in a single sequence have been proposed. For example, a study [30] addressed the ”rare item problem” by proposing to define a minimum support threshold for each item rather than using the same threshold for all items. In another study, an algorithm named *MTKPP* was proposed to discover the *k* periodic patterns that are the most frequent in a sequence [4]. To avoid the problems that comes with finding frequent patterns appearing periodically in a long sequence, a study [31] introduced a new interestingness measure to discover periodic-frequent patterns that occur almost periodically in a sequence. Considering the interestingness of mining periodic patterns but also to ignore unimportant events between important ones, and consider variable length patterns, an algorithm was proposed which lets the user determine the period value and the number of intermediate events that can be ignored [38].

However, a drawback of previous studies is that the maximum periodicity constraint is very strict. If a pattern appears regularly in a database but appear a single time with a time interval larger than the maximum periodicity threshold, it is discarded. Thus more flexible measures are needed [18].

Although many algorithms have been designed to find periodic patterns in a sequence, they are unable to find periodic patterns in multiple sequences. An algorithm was proposed [32] that take spatial locations of events into consideration and the concept of time lag for periodic pattern discovery. It makes use of a user-specified sliding-window to find periodic patterns in fixed order or non-fixed order. A user-specified period *p*, which is more like a maximum periodicity threshold mentioned above, is also used. As a result, it also faces the problem of setting this threshold properly. Moreover, a new data structure named ALT was presented [7] to allow users to avoid specifying periods in advance. But this study mainly aims at finding periodic patterns in discrete data sequences, rather than in symbolic sequence datasets, as considered in this paper. In recent years, an algorithm named PHUSPM [8] was designed to discover patterns in multiple symbolic sequences. However, that algorithm considers multiple sequences as a sequence, and then simply applies the same periodicity measures that were designed for mining periodic patterns in a single sequence. As a result, the algorithm does not consider whether a pattern is periodic in each sequence. Thus, the algorithm is unable to find patterns such that several customers periodically buy some products every week. This paper addresses this issue by proposing a more general model and algorithm for mining periodic patterns that are periodic in multiple sequences.

Note that this paper is about mining periodic patterns in symbolic data (sequences of transactions). Differently from this paper and the above related work, some studies have considered mining periodic patterns in numeric data (time series). The techniques for mining periodic patterns in numeric data are different than those for symbolic data, and rely for example, on the use of the Fourier transform to detect periodicity [12].

3. Definitions and problem statement

This section is divided into three parts. The first subsection presents the traditional problem of discovering periodic patterns in a single sequence. Then, the following subsection extends the traditional problem with a novel measure called the periodic standard deviation to filter non interesting periodic patterns in a sequence. Finally, the last subsection generalizes

$$\langle (a,b,c),(b,d),(a,b,e),(c),(a,b,d,e),(a,c),(b,c),(b,e)\rangle$$

Fig. 2. An example sequence.

that problem to mine patterns that are periodic in multiple sequences of a sequence database using a novel measure called the sequence periodic ratio.

3.1. The traditional problem of finding periodic patterns in a single sequence

The traditional problem of periodic pattern mining is defined for a single sequence [18]. A sequence is formally defined as follows.

Definition 1. Let there be a set of items I (symbols). An itemset X is a subset of I, that is $X \subseteq I$. An itemset containing k items is said to be a k-itemset. A sequence s is an ordered list of itemsets $s = \langle T_1, T_2, \dots T_m \rangle$, where $T_j \subseteq I$ $(1 \leq j \leq m)$, j is the transaction identifier of T_j, and T_j is said to be a transaction.

For example, consider a set of items $I = \{a, b, c, d, e\}$ representing products sold in a retail store, and the sequence of transactions shown in Fig. 2. This sequence contains eight itemsets (transactions). The first itemset contains three items (a, b and c). It is thus a 3-itemset.

To find periodic patterns in a single sequence, the concept of periods of an itemset was introduced, which is defined as follows [18].

Definition 2. A sequence $s_a = \langle A_1, A_2, \dots, A_k \rangle$ is said to be a subsequence of a sequence $s_b = \langle B_1, B_2, \dots, B_l \rangle$ iff there exist integers $1 \leq i1 < i2 < \dots < ik \leq m$ such that $A_1 \subseteq B_{i1}, A_2 \subseteq B_{i2}, \dots, A_k \subseteq B_{il}$ (denoted as $s_a \sqsubseteq s_b$).

For example, the sequence $\langle (a, b), (a) \rangle$ is a subsequence of the sequence depicted in Fig. 2.

Definition 3. Consider a sequence s and an itemset X. Let $TR(X, s) = \langle T_{g_1}, T_{g_2}, \dots, T_{g_k} \rangle \sqsubseteq s$ be the ordered set of transactions in which itemset X occurs in sequence s. Two transactions T_x and T_y in s are said to be *consecutive with respect to* X if there does not exist a transaction $T_z \in s$ such that $x < z < y$ and $X \subseteq T_z$. The period of two consecutive transactions T_x and T_y for a pattern X is $per(T_x, T_y) = y - x$. The periods of X in a sequence s are $pr(X, s) = \{per_1, per_2, \dots, per_{k+1}\}$ where $per_1 = g_1 - g_0$, $per_2 = g_2 - g_1, \dots per_{k+1} = g_{k+1} - g_k$, and $g_0 = 0$ and $g_{k+1} = n$, respectively.

For example, the itemset $\{a, b\}$ occurs in transactions T_0, T_2 and T_4 of the sequence s shown in Fig. 2. Thus, $TR(\{a, b\}, s_1) = \{T_0, T_2, T_4\}$ and the periods of pattern $\{a, b\}$ are $pr(\{a, b\}, s) = \{0, 2, 2, 3\}$.

The most widely used measure to assess the periodicity of a pattern in a single sequence is the maximum periodicity [18].

Definition 4. The maximum periodicity of an itemset X in a sequence s is defined as $maxPr(X, s) = argmax(pr(X, s))$.

In previous work, a pattern is deemed periodic if its maximum period is smaller than a user-defined *maxPr* threshold, and it appears frequently. Formally, the traditional task of mining periodic patterns in a sequence is defined as follows.

Definition 5. The suppport of an itemset X in a sequence s is the number of transactions containing X in s, that is $sup(X, s) = |TR(X, s)|$.

Definition 6. Let there be a sequence s, and two user-defined thresholds, namely the minimum support threshold *minSup* and the maximum periodicity threshold *maxPr*. The traditional problem of mining periodic patterns in a sequence s is to find each itemset X such that $sup(X, s) \geq minSup$ and $maxPr(X, s) \leq maxPr$.

For example, consider the sequence s shown in Fig. 2 and that $maxPr = 3$ and $minSup = 3$. The itemset $\{a, b\}$ is periodic since its periods in this sequence are $pr(\{a, b\}, s_1) = \{1, 2, 2, 3\}$, its maximum period is $maxPr(\{a, b\}, s) = max\{1, 2, 2, 3\} = 3 \leq maxPr$ and $sup(\{a, b\}, s) = 3 \geq minSup$.

In previous work, algorithms for the traditional problem have used the following property of the maximum periodicity measure to reduce the search space.

Theorem 1. *For two itemsets $X \subseteq X'$ in a sequence s, $maxPr(X, s) \leq maxPr(X', s)$ [18].*

For example, consider the sequence s shown in Fig. 2 and that $maxPr = 2$. The itemset $\{a, b\}$ is not periodic in sequence s_1 since its maximum period is $maxPr(\{a, b\}, s) = 3 \leq maxPr$. Thus, all supersets of $\{a, b\}$ such as $\{a, b, c\}$ are not periodic patterns in that sequence and do not need to be considered.

3.2. Extending the traditional problem with standard deviation

Although the problem of periodic pattern mining is useful, a limitation is that if *maxPr* is set to a small value, patterns may be discarded if they only have greater than *maxPr*, while if *maxPr* is set to a large value, patterns having periods that vary greatly may be considered as periodic. For example, consider the sequence s of Fig. 2. If $minSup = 2$ and $maxPr = 2$,

Table 2
An example sequence database.

Sequence_id	Sequence
0	$\langle(a, b, e), (a, b, e), (a, d), (a, e), (a, b, c)\rangle$
1	$\langle(c), (a, b, c, e), (c, d), (a, b, c, e), (a, b, d)\rangle$
2	$\langle(b, c), (a, b), (a, c, d), (a, c), (a, b)\rangle$
3	$\langle(a, b, d, e), (a, b, e), (a, b, c), (a, b, d, e), (a, b)\rangle$

the itemset $\{a, b\}$ is not periodic because its periods are {1,1,3,0} and $maxPr(\{a, b\}) = 3 > maxPr$. Hence, although the itemset $\{a, b\}$ always appear with a small period, it is discarded because it has a single period greater than *maxPr*. If we increase the *maxPr* threshold to $maxPr = 3$, then the itemset $\{a, b\}$ is considered periodic. However, the itemset $\{a, b, e\}$ also become periodic because its periods are {1,1,3} and $maxPr(\{a, b, e\}) = 3 \leq maxPr$. But this itemset actually only appear at the beginning of sequence *s*. It can thus be argued that it should not be considered as a periodic pattern since it has periods that vary greatly. To avoid finding patterns with periods that vary too much, this paper proposes to consider the standard deviation of periods.

Definition 7. The standard deviation of the periods of an itemset *X* in a sequence *s* is denoted as *stanDev*(*X*, *s*).

For example, consider the sequence *s* of Fig. 2. The periods of itemset $\{a, b\}$ in *s* are $pr(\{a, b\}, s) = \{1, 2, 2, 3\}$. The average period is $avgPr(\{a, b\}, s) = (1 + 2 + 2 + 3)/4 = 2$. The standard deviation is $stanDev(\{a, b\}, s) = \sqrt{[(1-2)^2 + (2-2)^2 + (2-2)^2 + (3-2)^2]/4}$.

This paper proposes to use the standard deviation as it is a statistical measure that is commonly used to evaluate the amount of dispersion or variation among a set of values. A small standard deviation means that values are closer to the average, and thus that the periods of a pattern should be more stable over time. Note that the algorithms designed in this paper could be easily modified to use other measures such as the variance to measure variations of periods. But an advantage of using the standard deviation instead of the variance is that the former is expressed using the same units as values. The average could also be used without much modifications to the proposed algorithms.

Based on the above definition, we consider that an itemset *X* is periodic in a sequence *s* if it meets the following conditions.

Definition 8. Let there be three user-specified thresholds *maxPer, minSup* and *maxStd*. An itemset *X* is periodic in a sequence *s* if $maxPer(X, s) \leq maxPr$, $sup(X, s) \geq minSup$ and $stanDev(X, s) \leq maxStd$.

For example, the itemset $\{a, b\}$ is periodic in sequence *s* for $maxStd = 2.0$, since $stanDev(\{a, b\}, s) = 0.707$.

3.3. Proposed problem of finding periodic patterns in multiple sequences

The previous definitions can be used to identify periodic patterns in a single sequence. Finding periodic patterns in a sequence has several applications such as to analyze a sequence of transactions made by a customer to find the sets of items that he purchases periodically. The following paragraphs explain how it is extended to discover periodic frequent patterns common to multiple sequences using a novel measure called the sequence periodic ratio. Analyzing multiple sequences is useful for example, to discover periodic patterns that are common to many customers.

Definition 9. A sequence database *D* is an ordered set of *n* sequences, denoted as $D = \langle s_1, s_2, \ldots, s_n\rangle$. The sequence s_i in *D* is said to be the *i*-th sequence of *D*, and its sequence identifier is said to be *i*.

For example, consider the database of Table 2, containing four sequences, which will be used as running example. The first sequence (s_1) contains five itemsets. The first itemset contains two items (*a* and *b*). It is thus a 2-itemset. The sequence $\langle(a, b), (a)\rangle$ is a subsequence of s_1.

Definition 10. The number of sequences where an itemset *X* is periodic in a sequence database *D* is denoted and defined as $numSeq(X) = |\{s | maxPer(X, s) \leq maxPr \wedge sup(X, s) \geq minSup \wedge stanDev(X, s) \leq maxStd \wedge s \in D\}|$. The sequence periodic ratio of *X* in *D* is defined as $ra(X) = numSeq(X)/|D|$, where $|D|$ is the number of sequences in *D*.

For example, the number of sequences where $\{a, b\}$ is periodic is $numSeq(\{a, b\}) = 3$ (it is periodic in s_1, s_2, and s_4). The total number of sequences is $|D| = 4$. Thus, the sequence periodic ratio of $\{a, b\}$ is $ra(\{a, b\}) = 3/4 = 0.75$.

Problem statement. Let there be a sequence database *D*, and four user-defined thresholds, namely the minimum support threshold *minSup*, maximum periodicity threshold *maxPr*, maximum standard deviation threshold *maxStd*, and minimum sequence periodic ratio threshold *minRa*. An itemset *X* is a Periodic Frequent Pattern (PFPS) in *D* if $ra(X) \geq minRa$. The problem of mining periodic patterns common to multiple sequences is to find all PFPS.

For example, Table 3 shows the PFPS found for different thresholds values. The first line uses $minSup = 2$, $maxPr = 3$, $maxStd = 5.0$ and $minRa = 0.3$, while the following lines change one parameter with respect to the first line (in bold). It can be seen that each parameter is useful to reduce the number of patterns, and thus the parameters provide a lot of flexibility to the user to select the desired patterns.

Table 3
Patterns found for different threshold values.

No.	*minSup*	*maxPr*	*maxStd*	*minRa*	Patterns found
1	2	3	1.0	0.6	{*a*}, {*e*}, {*a*, *e*}
2	**3**	3	1.0	0.6	{*a*}
3	2	**1**	1.0	0.6	{*a*}
4	2	3	**1.5**	0.6	{*a*}, {*b*}, {*e*}, {*a*, *b*}, {*a*, *e*},
5	2	3	1.0	**0.4**	{*a*}, {*b*}, {*c*}, {*e*}, {*a*, *b*}, {*a*, *e*}, {*b*, *e*}, {*a*, *b*, *e*}

The traditional problem of mining periodic patterns [18] is a special case of the proposed problem where the database contains a single sequence, $maxStd = \infty$ and $minRa = 0$. The proposed problem can be viewed as more challenging than the traditional problem because multiple sequences must be compared while exploring the search space of itemsets and two new measures must be taken into account to assess the periodicity in multiple sequences. Traditional algorithms for periodic pattern mining cannot be directly applied to the proposed problem because their search procedures and data structures are designed for handling a single sequence. The next section presents two efficient algorithms for the proposed problem.

4. The proposed algorithms

This section introduces two efficient algorithms to mine PFPS, named MPFPS$_{BFS}$ (Mining Periodic Frequent Pattern common to multiple Sequences using a breadth first search), and MPFPS$_{DFS}$ (using a depth first search). These two algorithms are based on two novel pruning properties and a novel data structure called PFPS-list. This section first presents the novel properties and data structure and then describes the two proposed algorithms.

4.1. The pruning strategies

To find periodic patterns efficiently in multiple sequences, it is necessary to find a way of reducing the search space. For the traditional problem of mining periodic patterns in a single sequence, Theorem 1 can be used because the maximum periodicity measure is anti-monotone. For the proposed problem, the *ra* measure is neither anti-monotone nor monotone and thus cannot be directly used to reduce the search space.

Theorem 2. *For two itemsets* $X \subseteq X'$, *either* $ra(X) < ra(X')$, $ra(X) = ra(X')$ *or* $ra(X) > ra(X')$.

Proof. The proof is made by giving an example for each of the three cases. Consider the database of Table 2 and that $minSup = 2$, $maxPr = 3$ and $maxStd = 0.8$. It can be found that $ra(\{b\}) = 0 < ra(\{b, e\}) = 0.25$, $ra(\{e\}) = 0.25 = ra(\{a, e\})$ and $ra(\{e\}) = 0.25 > ra(\{d, e\}) = 0$. □

To be able to reduce the search space, this paper introduces a new measure called *boundRa* that is an upper-bound on the *ra* measure and is monotone.

Definition 11. Given two user-specified thresholds *maxPer* and *minSup*, an itemset *X* is a candidate in a sequence *s* if $maxPr(X, s) \leq maxPr$ and $sup(X, s) \geq minSup$. The number of sequences where an itemset *X* is a candidate in a sequence database *D* is denoted as *numCand*(*X*), and defined as $numCand(X) = |\{s | maxPr(X, s) \leq maxPr \wedge sup(X, s) \geq minSup \wedge s \in D\}|$. The *boundRa* of *X* in *D* is defined as $boundRa(X) = numCand(X)/|D|$.

The measure *boundRa* has two important properties that are useful to reduce the search space.

Theorem 3. *For an itemset X,* $boundRa(X) \geq ra(X)$.

Proof. For an itemset *X*:

$$\begin{aligned} ra(X) &= numSeq(X)/|D| \\ &= |\{s | maxPr(X, s) \leq maxPr \wedge sup(X, s) \geq minSup \wedge stanDev(X, s) \leq maxStd \wedge s \in D\}|/|D| \\ &\leq |\{s | maxPr(X, s) \leq maxPr \wedge sup(X, s) \geq minSup \wedge s \in D\}|/|D| \\ &= numCand(X)/|D| \\ &= boundRa(X) \end{aligned}$$

□

Theorem 4. *For two itemsets* $X' \subseteq X$, $boundRa(X') \geq boundRa(X)$.

Proof. Because for any sequence s:

$$\begin{aligned} X' \subseteq X &\Rightarrow maxPr(X', s) \leq maxPr(X, s) && \text{(by Theorem 1)} \\ &\Rightarrow sup(X', s) \geq sup(X, s) && \text{([18])} \end{aligned}$$

Therefore:

$$\begin{aligned} boundRa(X') &= numCand(X')/|D| \\ &= |\{s|maxPr(X', s) \leq maxPr \wedge sup(X', s) \geq minSup \wedge s \in D\}|/|D| \\ &\geq |\{s|maxPr(X, s) \leq maxPr \wedge sup(X, s) \geq minSup \wedge s \in D\}|/|D| \\ &= numCand(X)/|D| \\ &= boundRa(X) \end{aligned}$$

□

Based on Theorem 4, if it is found that the *boundRa* of an itemset is smaller than the *minRa* threshold, then this itemset and its supersets are not PFPS and thus do not need to be considered.

Theorem 5. *If* $boundRa(X') < minRa$ *for an itemset* X'*, then* X' *and any superset* $X \supset X'$ *are not PFPS.*

Proof.

$$boundRa(X') < minRa \Rightarrow ra(X') < minRa \qquad \text{(by Theorem 3)}$$

Thus X is not a PFPS. Moreover:

$$\begin{aligned} X \supset X' &\Rightarrow boundRa(X) \leq boundRa(X') && \text{(by Theorem 4)} \\ &\Rightarrow ra(X) < minRa && \text{(by Theorem 3)} \end{aligned}$$

Hence, any superset X of X' is not a PFPS. □

A second pruning theorem is also introduced, which is a variation of the above Theorem.

Theorem 6. *For an itemset* X'*, if* $\exists X'' \subset X'$ *such that* $boundRa(X'') < minRa$*,* X' *is not a PFPS. Moreover, for any itemset* $X \supset X'$*,* X *is not a PFPS.*

Proof.

$$\begin{aligned} X'' \subset X' \subset X &\Rightarrow boundRa(X) \leq boundRa(X') \leq boundRa(X'') && \text{(by Theorem 4)} \\ &\Rightarrow boundRa(X) \leq boundRa(X') < minRa && (\text{since } boundRa(X'') < minRa) \end{aligned}$$

Hence, $boundRa(X) < boundRa(X') < minRa$ and itemsets X and X' are not PFPS. □

4.2. The PFPS-list data structure

The proposed algorithms explore the search space of itemsets, which contains $2^{|I|}$ itemsets. Both algorithms start from single items, and recursively append an item to each itemset to generate a larger itemset, following the $\succ$ order. The two algorithms reduce the search space by exploiting the fact that *boundRa* is an upper-bound on the *ra* measure and satisfies the *downward closure property* (Theorem 4). To calculate all the measures to evaluate patterns without having to repeatedly scan the database, the proposed algorithms utilize a novel data structure called PFPS-list. A PFPS-list is created for each itemset that is visited in the search space.

Definition 12. Let *sequences*(*X*) be the set of sequences containing *X*, that is $sequences(X) = \{s|sup(X, s) > 0 \wedge s \in D\}$, ordered by ascending sequence identifiers.

For example, consider the sequence database of Table 2. The sequences containing itemset $\{a\}$ are $sequences(\{a\}) = \{s_1, s_2, s_3, s_4\}$.

Definition 13. The PFPS-list *LX* of an itemset *X* contains three fields. The *i-set* field is defined as $LX.i-set = X$. The *sid-list* field is defined as a list $LX.sidlist = \{v_1, v_2 \ldots v_w\}$ where $w = |sequences(X)|$ and v_i $(1 \leq i \leq w)$ is the sequence identifier of the *i*-th sequence in *sequences*(*X*). Thus, this field contains the list of identifiers of sequences containing *X*. The *tidlist-list* field is defined as a list $LX.tidlist-list = \{Z_1, Z_2 \ldots Z_w\}$ where $w = |sequences(X)|$. Let s_i be the *i*-th sequence in *sequences*(*X*) $(1 \leq i \leq w)$. The value Z_i is defined as $Z_i = \{z|X \subseteq T_z \wedge \langle T_z \rangle \sqsubseteq s_i\}$. In other words, this field stores the list of identifiers of transactions containing *X*, for each sequence containing *X*.

For example, the PFPS-lists of the itemsets $\{a\}$ and $\{e\}$ are shown in Tables 4 and 5, respectively.

The information stored in the PFPS-list of an itemset *X* allows to quickly find the transactions and sequences where it appears, and thus it provides all the required information to determine if *X* is periodic without scanning the database. In particular, the field $sid-list$ can be used to retrieve *sequences*(*X*), that is all sequences containing *X*. Then, using this information, the list of transactions for each sequence *s* containing *X* can be obtained from the field $tidlist-list$. This allows to calculate the periods of *X* in *s*, that is *pr*(*X*, *s*), which then allows to calculate *sup*(*X*, *s*), *maxpr*(*X*, *s*) and *stanDev*(*X*, *s*). By calculating these values for all sequences containing *X*, *boundRa*(*X*) and *ra*(*X*) can also be obtained.

Table 4
The PFPS-list of itemset {*a*}.

i-set	{*a*}
sid-list	{0,1,2,3}
tidlist-list	[{0,1,2,3,4}, {1,3,4},{1,2,3,4}, {0,1,2,3,4}]

Table 5
The PFPS-list of itemset {*e*}.

i-set	{*e*}
sid-list	{0,1,3}
tidlist-list	[{0,1,3}, {1,3,4}, {0,1,3}]

The proposed algorithms scan the database once to construct the PFPS-list of each item. Then, the PFPS-list of any itemset containing more than one item is obtained by performing an intersection operation on the PFPS-lists of two of its subsets. In other words, it is unnecessary to scan the database to create the PFPS-list of an itemset having more than one item. The intersection operation is performed by the *Intersect* procedure (Algorithm 1). It is based on the concept of extension of an itemset.

Algorithm 1: The Intersect procedure.

```
Input: LPx and LPy: the PFPS-lists of two extensions Px and Py of an itemset
output: the PFPS-list LPxy of itemset Pxy
1  $LPxy.i\text{-}set \leftarrow Px \cup \{y\}$; $LPxy.tidlist\text{-}list \leftarrow \emptyset$; $LPxy.sid\text{-}list \leftarrow \emptyset$;
2  foreach sequence identifier $sid \in LPx.sid\text{-}list$ such that $sid \in LPy.sid\text{-}list$ do
3      $tidListSidPx \leftarrow$ the tid list of sid in $LPx.tidlist\text{-}list$;
4      $tidListSidPy \leftarrow$ the tid list of sid in $LPy.tidlist\text{-}list$;
5      $tidListSidPxy \leftarrow tidListSiPx \cap tidListSiPy$;
6      if $tidListSidPxy \neq \emptyset$ then
7          $LPxy.sid\text{-}list.append(sid)$;$LPxy.tidlist\text{-}list.append(tidListSidPxy)$;
8      end
9  end
10 return LPxy;
```

Definition 14. The extension of an itemset *P* with an item *z* is defined as $Pz = P \cup \{z\}$.

The *Intersect* procedure takes as input the PFPS-lists of two itemsets *Px* and *Py*, denoted as *LPx* and *LPy*, and outputs the PFPS-list of the itemset *Pxy*. The algorithm first initializes an empty PFPS-list *LPxy* for *Pxy* (line 1). Then, the algorithm performs a loop to consider each sequence that appears in both the PFPS-lists of *Px* and *Py*. For each such sequence, let *sid* be the sequence identifier representing that sequence. That sequence identifier is used to retrieve the lists of identifiers of transactions containing *Px* and *Py* in that sequence, denoted as *tidListSidPx* and *tidListSidPy*, respectively (line 3 and 4). These two lists are then intersected to obtain the list of identifiers of transactions containing *Pxy* in that sequence, named *tidListSidPxy* (line 5). If that latter list is not empty, *sid* is added to the PFPS-list of *Pxy* as well as the list of transactions *tidListSidPxy*. The procedure returns the PFPS-list of *Pxy*, which is obtained without scanning the database.

For example, by applying the *Intersect* procedure on the PFPS-lists of itemsets {*a*} and {*e*}, depicted in Tables 4 and 5, the PFPS-list of itemset {*a, e*} is obtained, shown in Table 6.

4.3. The MPFPS$_{\text{BFS}}$ algorithm

The first proposed algorithm is named MPFPS$_{BFS}$ (Algorithm 2) and performs a breadth-first search to discover all PFPS. It takes as input a database with multiple sequences and the *maxStd, minRa, maxPr*, and *minSup* thresholds. MPFPS$_{BFS}$ outputs

Table 6
The PFPS-list of itemset {*a, e*}.

i-set	{*a, e*}
sid-list	{0,1,3}
tidlist-list	[{0,1,3}, {1,3}, {0,1,3}]

all the PFPS. The algorithm first read the database to calculate *sup*({*i*}, *s*), *pr*({*i*}, *s*), *maxpr*({*i*}, *s*) and *stanDev*({*i*}, *s*) for each item *i* and sequence *s* (line 1).

Algorithm 2: The MPFPS$_{BFS}$ algorithm.

```
Input: $D$: a database with multiple sequences, $maxStd, minRa, maxPr, minSup$: the thresholds.
output: the set of periodic frequent patterns (PFPS).
1  Scan each sequence $s \in D$ to calculate $sup(\{i\}, s)$, $pr(\{i\}, s)$, $maxpr(\{i\}, s)$ and $stanDev(\{i\}, s)$ for each item $i \in I$;
2  foreach item $i \in I$ do
3      $numSeq(\{i\}) \leftarrow |\{s | maxpr(\{i\}, s) \leq maxPr \wedge stanDev(\{i\}, s) \leq maxStd \wedge sup(\{i\}, s) \geq minSup \wedge s \in D\}|$;
4      $ra(\{i\}) \leftarrow numSeq(\{i\})/|D|$;
5      if $ra(\{i\}) \geq minRa$ then output $\{i\}$;
6      $numCand(\{i\}) \leftarrow |\{s | maxpr(\{i\}, s) \leq maxPr \wedge sup(\{i\}, s) \geq minSup \wedge s \in D\}|$;
7      $boundRa(\{i\}) \leftarrow numCand(\{i\})/|D|$;
8  end
9  $boundPFPS \leftarrow$ {PFPS-list of item $i | i \in I \wedge boundRa(\{i\}) \geq minRa$};
10 while $|boundPFPS| \geq 2$ do
11     GenerateItemsets ($boundPFPS, minSup, maxPr, maxStd, minRa, D$);
12 end
```

Then, MPFPS$_{BFS}$ checks if each item *i* is periodic in each sequence of the database (line 3 to 5). For an item *i* appearing in a sequence *s*, if *sup*({*i*}, *s*) ≥ *minSup, maxpr*({*i*}, *s*) ≤ *maxPr* and *stanDev*({*i*}, *s*) < *maxStd*, then *i* is said to be periodic in that sequence. Then, the algorithm calculates the sequence periodic ratio of item *i* by dividing the number of sequences where *i* is periodic by the total number of sequences. If this value is not less than *minRa, i* is a PFPS and it is output (line 5). Also, *boundRa* of {*i*} is calculated to prune the search space (line 6 to 7) and the PFPS-list of each single item *i* such that *boundRa*({*i*}) ≥ *minRa* is stored in a set *boundPFPS* (line 8). Then, the algorithm performs a breadth-first search to find PFPS. This is done by a while loop (line 10 to 12), where the procedure *GenerateItemsets* is called repeatedly to generate larger itemsets until no itemsets can be generated. The *GenerateItemsets* procedure is first called with *boundPFPS* which contains all 1-itemsets having a *boundRa* value no less than *minRa*. The procedure combines pairs of these 1-itemsets to obtain 2-itemsets having a *boundRa* value no less than *minRa*, which are then stored in *boundPFPS*. At the same time, PFPS containing two items are identified and output. Generally, each time that the *GenerateItemsets* procedure is called, it combines *k*-itemsets ($k \geq 1$) to generate ($k+1$)-itemsets. The while loop stops when *boundPFPS* contains less than two itemsets.

The *GenerateItemsets* procedure (Algorithm 3) takes as input a set of PFPS-lists of *k*-itemsets ($k \geq 1$) called *boundPFPS*, the *minSup, maxPr, maxStd, minRa* thresholds and the database *D*. The procedure outputs the set of PFPS of length *k*. The procedure first initializes a variable $boundPFPS_{k+1}$ to store ($k+1$)-itemsets having *boundRa* values no less than *minRa* (line 1). Then, all the itemsets corresponding to the PFPS-lists in *boundPFPS* are put in a variable *itemsets* (line 2). Thereafter, loops are performed to combine pairs of *k*-itemsets to generate ($k+1$)-itemsets. Let there be any total order $\succ$ on items. Two itemsets are combined to obtain a ($k+1$)-itemset *Pxy* if they are of the form *Px* and *Py*, that is if they share a same prefix *P* of $k-1$ items, and if $x \succ y$. In that case, the PFPS-list of the itemset *Pxy* is created by calling the *Intersect* procedure with the PFPS-lists of *Px* and *Py* (line 5). Then, the algorithm checks if all the subsets of *Pxy* having *k* items are PFPS (appear in the set *itemsets*) (line 7). If this condition is not met, *Pxy* and its supersets are not PFPS according to the subset pruning property (Theorem 6). Otherwise, the PFPS-list of *Pxy* is read once to calculate *maxpr*(*Pxy, s*) and *sup*(*Pxy, s*) for each sequence *s* appearing in the PFPS-list of *Pxy* (line 8). Based on these values, *boundRa*(*Pxy*) is calculated (line 9). If this value is less than *minRa, Pxy* and all its supersets cannot be PFPS by Theorem 5 (line 10). Otherwise, the PFPS-list of *Pxy* is added to $boundPFPS_{k+1}$ since it could be used to generate larger PFPS (line 11). Then, *ra*(*Pxy*) is calculated by reading the PFPS-list of *Pxy* once (line 12 to 13). If this value is no less than *minRa, Pxy* is a PFPS and is output (line 14). Finally, after combining all *k*-itemsets that can be combined, $boundPFPS_{k+1}$ is copied into *boundPFPS* (line 19) to prepare for generating the ($k+2$)-itemsets in the next call of the *GenerateItemsets* procedure.

The algorithm is complete since it only reduces the search space using Theorems 5 and 6, thus only eliminating itemsets that are not PFPS. The algorithm is correct since it calculates the *ra* value of each other itemset using its PFPS-list and only output those having a value no less than *minRa*.

4.3.1. A detailled example

Consider the example database of Table 2 and that $minSup = 2, maxPr = 3,$ $maxStd = 1.0$ and $minRa = 0.6$. The main procedure of MPFPS$_{BFS}$ (Algorithm 2) first processes single items. Consider the item *a*. By scanning the database, it is found that *pr*({*a*}, s_1) = [1,1,1,1,1,0], *pr*({*a*}, s_2) = [2,2,1,0], *pr*({*a*}, s_3) = [2,1,1,1,0], *pr*({*a*}, s_4) = [1,1,1,1,1,0], $sup(\{a\}, s_1) = 5,$ $sup(\{a\}, s_2) = 3,$ $sup(\{a\}, s_3) = 4,$ $sup(\{a\}, s_4) = 5,$ $maxpr(\{a\}, s_1) = 1,$ $maxpr(\{a\}, s_2) = 2,$ $maxpr(\{a\}, s_3) = 2,$ $maxpr(\{a\}, s_4) = 1,$ $stanDev(\{a\}, s_1) = 0.152,$ $stanDev(\{a\}, s_2) = 0.256,$ $stanDev(\{a\}, s_3) = 0.632,$ and $stanDev(\{a\}, s_4) = 0.152$. As *sup*({*a*}, s_1) > *minSup, maxpr*({*a*}, s_1) < *maxPr* and *stanDev*({*a*}, s_1) < *maxStd*, item {*a*} is periodic in sequence s_1. Sim-

Algorithm 3: The *GenerateItemsets* procedure.

```
Input: boundPFPS: a set of PFPS-lists of itemsets of length $k$, $minSup, maxPr, maxStd, minRa$: the thresholds, $D$: the database.
output: the set of periodic frequent patterns of length $k$.
1  $boundPFPS_{k+1} \leftarrow \emptyset$;
2  $itemsets \leftarrow \{LPt.i\text{-}set | LPt \in boundPFPS\}$;
3  foreach PFPS-list $LPx \in boundPFPS$ do
4      foreach PFPS-list $LPy \in boundPFPS$ such that $y \succ x$ and $LPx.i\text{-}set$ has the same prefix of $k-1$ items as $LPy.i\text{-}set$ do
5          $LPxy \leftarrow$ Intersect $(LPx, LPy)$;
6          $Pxy = LPxy.i\text{-}set$;
7          if $\forall SPxy \subset Pxy$ such that $|SPxy| = |Pxy| - 1, SPxy \in itemsets$ then
8              $numCand(Pxy) \leftarrow |\{s | maxpr(Pxy, s) \leq maxPr \wedge sup(Pxy, s) \geq minSup \wedge s \in D\}|$;
9              $boundRa(Pxy) \leftarrow numCand(Pxy)/|D|$;
10             if $boundRa(Pxy) \geq minRa$ then
11                 $boundPFPS_{k+1} \leftarrow boundPFPS_{k+1} \cup \{LPxy\}$;
12                 $numSeq(Pxy) \leftarrow |\{s | maxpr(Pxy, s) \leq maxPr \wedge stanDev(Pxy, s) \leq maxStd \wedge sup(Pxy, s) \geq minSup \wedge s \in LPxy.sid\text{-}list\}|$;
13                 $ra(Pxy) \leftarrow numSeq(Pxy)/|D|$;
14                 if $ra(Pxy) \geq minRa$ then output $Pxy$;
15             end
16         end
17     end
18 end
19 $boundPFPS \leftarrow boundPFPS_{k+1}$;
```

ilarly, item {a} is also periodic in sequences s_2, s_3 and s_4. Hence, {a} is periodic in *numSeq*({a}) = 4 sequences and the ratio *ra*({a}) = 1 ≥ *minRa*. Thus, {a} is a PFPS and it is output. Then, *boundRa*({a}) is calculated. It is found that *numCand*({a}) = 4 and *boundRa*({a}) = 1. The other items are processed in the same way and it is found that {e} is also a PFPS. Then, all the PFPS-lists of items having a *boundRa* value that is no less than *minRa* are created and added to the set *boundPFPS*. Thus, the PFPS-lists of items {a} and {e} are added to *boundPFPS*. For example, the PFPS-list of itemset {a} is created and added to *boundPFPS* since *boundRa*({a}) = 1 ≥ *minRa*. While the size of *boundRa* is not smaller than two, the *GenerateItemsets* procedure is called to generate larger itemsets and find all remaining PFPS (Algorithm 3).

Since itemsets {a} and {e} are both 1-itemsets, they share the same prefix. The *Intersect* procedure is thus called with the PFPS-lists of {a} and {e} to generate the PFPS-list of {a, e}. The *sid-list* of {a} is {0, 1, 2, 3} and that of {e} is {0, 1, 3}. Thus, the *sid-list* of {a, e} is {0, 1, 3}. For the first sequence, {a}'s tidlist is {0, 1, 2, 3, 4}, {e}'s tidlist is {0, 1, 3}, and their intersection is {0, 1, 3}. Thus, for the first sequence, the sequence identifier 0 is added to the *sid-list* of {a, e} and the intersection {0, 1, 3} is added to the *tidlist-list* of {a, e}. The rest of the PFPS-list of {a, e} is constructed in the same way for the other sequences. The PFPS-list of {a, e} is shown in Table 6. Then, the algorithm checks if all subsets of {a, e}, that is {a} and {e}, are contained in the set *boundPFPS*. Since they are, the *boundRa* value of {a, e} is calculated using its PFPS-list. Since it is found that *boundRa*({a, e}) = 0.6 ≥ *minRa*, the itemset {a, e} and its supersets may be PFPS and the PFPS-list of {a, e} is added to $boundPFPS_2$. Hence, the sequences in which {a, e} is periodic are found and *ra*({a, e}) is calculated. Since *numSeq*({a, e}) = 3, the ratio *ra*({a, e}) = 0.75 ≥ *minRa*, and thus {a, e} is a PFPS and it is output.

4.3.2. Complexity

The complexity of the $MPFPS_{BFS}$ algorithm is analyzed as follows. The algorithm first scans the database to calculate *sup*({i}, s), *pr*({i}, s), *maxpr*({i}, s) and *stanDev*({i}, s) for each item $i \in I$. The time cost of this step is $\mathcal{O}(n \times w)$ where w is the average number of transactions per sequence. After that the algorithm performs a loop over each item to check if it is a PFPS and determine if the item should be added to *boundPFPS*. The time cost of this step is $\mathcal{O}(|I|)$. Then, the algorithm does a breadth-first search by calling *GenerateItemsets* until no itemsets can be generated. A call to *GenerateItemsets* compares all pairs of k-itemsets in *boundPFPS* to generate $(k+1)$-itemsets. If *boundPFPS* contains l itemsets, then $(l \times (l-1))/2$ pairs of itemsets are combined to generate $(k+1)$-itemsets. For each pair of itemsets *Px* and *Py*, the *Intersect* procedure is called, which is implemented as a two-way search. It scans the PFPS-lists of *Px* and *Py* once to build the PFPS-list of *Pxy*. Thus, the time cost of the *Intersect* procedure is at most $\mathcal{O}(g+h)$ where g and h are the size of the PFPS-lists of *Px* and *Py*, respectively. In the worst case, g and h are equal to the number of transactions $n \times w$ in the database and thus this process is $\mathcal{O}(n \times w)$. Then, the PFPS-list of *Pxy* is scanned to calculate *boundRa*(*Pxy*) and *ra*(*Pxy*), which has a time cost of at most $\mathcal{O}(n \times w)$. The algorithm explores the search space of itemsets using the above process. In the worst case, $2^{|I|} - 1$ itemsets are generated. Depending on how the algorithm's parameters are set, the pruning properties can reduce the search space more or less. Hence, the overall time cost of the MPFPS$_{BFS}$ algorithm is $\mathcal{O}(n \times w + |I|)$ to evaluate the 1-itemsets, and then $\mathcal{O}(n \times w)$ for each itemset considered in the search space.

In terms of space cost, the algorithm creates a PFPS-list for each itemset. During the breadth-first search, at any moment, the algorithm only needs to keep the PFPS-lists of k-itemsets and $(k+1)$-itemsets in memory ($k \geq 1$). In the worst case, a PFPS-list takes $\mathcal{O}(n)$ space to store the $sid-list$ and $\mathcal{O}(n \times w)$ to store $tidlist-list$. Hence, the overall space cost is $\mathcal{O}(n + n \times w)$ for each itemset.

4.4. The MPFPS$_{\text{DFS}}$ algorithm

The second proposed algorithm is named MPFPS$_{DFS}$ (Algorithm 4) and perform a depth-first search to discover all PFPS. It takes as input a database with multiple sequences and the *maxStd, minRa, maxPr*, and *minSup* thresholds. MPFPS$_{DFS}$ outputs all the PFPS. It first scans the database once to calculate $sup(\{i\}, s)$, $pr(\{i\}, s)$, $maxpr(\{i\}, s)$ and $stanDev(\{i\}, s)$ for each item i and sequence s (line 1). Then, MPFPS$_{DFS}$ does a loop for each item. For an item i appearing in a sequence s, if $sup(\{i\}, s) \geq minSup$, $maxpr(\{i\}, s) \leq maxPr$ and $stanDev(\{i\}, s) < maxStd$, then i is said to be periodic in that sequence. Then, the algorithm calculates the sequence periodic ratio of item i by dividing the number of sequences where i is periodic by the total number of sequences (line 3 to 4). If this value is not less than *minRa*, i is a PFPS and it is output (line 5). Also, *boundRa* of $\{i\}$ is calculated to prune the search space (line 6 to 7) and the PFPS-list of each single item i such that $boundRa(\{i\}) \geq minRa$ is stored in a set *boundPFPS* (line 9). This set is sorted according to a total order $\succ$, defined as the ascending order of *boundRa* values. Then, the depth-first search of PFPS starts by calling the recursive *Search* procedure with *boundPFPS, minSup, maxPr, maxStd, minRa* and *D*. This procedure will only explore itemsets having a *boundRa* value no less than *minRa* because other itemsets are not PFPS according to Theorem 5.

Algorithm 4: The MPFPS$_{DFS}$ algorithm.

```
Input: D: a database with multiple sequences, maxStd, minRa, maxPr, minSup: the thresholds.
output: the set of periodic frequent patterns (PFPS).
1  Scan each sequence $s \in D$ to calculate $sup(\{i\},s)$, $pr(\{i\},s)$, $maxpr(\{i\},s)$ and $stanDev(\{i\},s)$ for each item $i \in I$;
2  foreach item $i \in I$ do
3      $numSeq(\{i\}) \leftarrow |\{s|maxpr(\{i\},s) \leq maxPr \wedge stanDev(\{i\},s) \leq maxStd \wedge sup(\{i\},s) \geq minSup \wedge s \in D\}|$;
4      $ra(\{i\}) \leftarrow numSeq(\{i\})/|D|$;
5      if $ra(Px) \geq minRa$ then output $Px$;
6      $numCand(\{i\}) \leftarrow |\{s|maxpr(\{i\},s) \leq maxPr \wedge sup(\{i\},s) \geq minSup \wedge s \in D\}|$;
7      $boundRa(\{i\}) \leftarrow numCand(\{i\})/|D|$;
8  end
9  $boundPFPS \leftarrow$ {PFPS-list of item $i|i \in I \wedge boundRa(\{i\}) \geq minRa$};
10 Sort $boundPFPS$ by the order $\succ$ of ascending $boundRa$ values;
11 Search ($boundPFPS$, $minSup$, $maxPr$, $maxStd$, $minRa$, $D$);
```

The *Search* procedure (Algorithm 5) takes as input PFPS-lists of extensions of an itemset P, the *minSup, maxPr, maxStd, minRa* thresholds and the database. An *extension* of an itemset P is an itemset that is obtained by appending an item z to P, and is denoted as Pz. When the procedure is first called, P is the empty set, and extensions of P are single items. The *Search* procedure performs loops to combine each pair of extensions Px and Py of P such that $y \succ x$, to generate an extension Pxy containing $|Px| + 1$ items (line 1 to 12). The PFPS-list of each such extension Pxy, denoted as $LPxy$ is created without scanning the database by applying the *Intersect* procedure (Algorithm 1) on the PFPS-lists of Px and Py (line 3). Then, the PFPS-list of Pxy is scanned to calculate $numCand(Pxy)$ and $boundRa(Pxy)$ (line 4 to 5). Then, if $boundRa(Pxy) < minRa$, Pxy is not a PFPS and all its supersets are not PFPS by Theorem 5 (line 6). Otherwise, the PFPS-list of Pxy is added to a set called *ExtensionsOfPx* that stores all PFPS-lists of extensions of Px having a *boundRa* value no less than *minRa* (line 7). Then, the ratio $ra(Pxy)$ is calculated to check if Pxy is a PFPS, and if yes, Pxy is output (line 8 to 10). Then, the *Search* procedure is recursively called with the PFPS-lists of PFPS that extend Px (*ExtensionsOfPx*) to explore their transitive extensions (line 13).

The algorithm is complete since it only reduces the search space using Theorems 5 and 6, thus only eliminating itemsets that are not PFPS. The algorithm is correct since it calculates the *ra* value of each other itemset using its PFPS-list and only output those having a value no less than *minRa*.

4.4.1. A Detailled Example

Consider the example database of Table 2 and that $minSup = 2$, $maxPr = 3$, $maxStd = 1.0$ and $minRa = 0.6$. The main procedure of MPFPS$_{DFS}$ (Algorithm 4) first processes single items. Consider the item a. By scanning the database, it is found that $pr(\{a\}, s_1) = [1,1,1,1,1,0]$, $pr(\{a\}, s_2) = [2,2,1,0]$, $pr(\{a\}, s_3) = [2,1,1,1,0]$, $pr(\{a\}, s_4) = [1,1,1,1,1,0]$, $maxpr(\{a\}, s_1) = 1$, $maxpr(\{a\}, s_2) = 2$, $maxpr(\{a\}, s_3) = 2$, $maxpr(\{a\}, s_4) = 1$, $stanDev(\{a\}, s_1) = 0.152$, $stanDev(\{a\}, s_2) = 0.256$, $stanDev(\{a\}, s_3) = 0.632$, and $stanDev(\{a\}, s_4) = 0.152$. As $maxpr(\{a\}, s_1) < maxPr$ and $stanDev(\{a\}, s_1) < maxStd$, item $\{a\}$ is periodic in sequence s_1. Similarly, it is also periodic in sequences s_2, s_3 and s_4. Hence, (a) is periodic in $numSeq(\{a\}) = 4$ sequences and the ratio

Algorithm 5: The *Search* procedure.

```
Input: $ExtensionsOfP$: a set of PFPS-lists of extensions of an itemset $P$, $minSup, maxPr, maxStd, minRa$: the thresholds, $D$: the database.
output: the set of periodic frequent patterns that extend $P$.
1  foreach PFPS-list $LPx \in ExtensionsOfP$ and $Px = LPx.i\text{-}set$ do
2      foreach PFPS-list $LPy \in ExtensionsOfP$ and $Py = LPy.i\text{-}set$ such that $y \succ x$ do
3          $LPxy \leftarrow$ Intersect $(LPx, LPy)$;
4          $numCand(Pxy) \leftarrow |\{s|maxpr(Pxy, s) \le maxPr \wedge sup(Pxy, s) \ge minSup \wedge s \in D\}|$;
5          $boundRa(Pxy) \leftarrow numCand(Pxy)/|D|$;
6          if $boundRa(Pxy) \ge minRa$ then
7              $ExtensionsOfPx \leftarrow ExtensionsOfPx \cup \{LPxy\}$;
8              $numSeq(Pxy) \leftarrow |\{s|maxpr(Pxy, s) \le maxPr \wedge stanDev(Pxy, s) \le maxStd \wedge sup(Pxy, s) \ge minSup \wedge s \in LPxy.sid\text{-}list\}|$;
9              $ra(Pxy) \leftarrow numSeq(Pxy)/|D|$;
10             if $ra(Pxy) \ge minRa$ then output $Pxy$;
11         end
12     end
13     Search $(ExtensionsOfPx, minSup, maxPr, maxStd, minRa, D)$;
14 end
```

$ra(\{a\}) = 1 \ge minRa$. Thus, {a} is a PFPS and it is output. Then, the same process is repeated for the other items and it is found that the items $\{a\}$ and $\{e\}$ are PFPS. Then, all the PFPS-lists of items having a *boundRa* value that is no less than *minRa* are created and added to the set *boundPFPS*. Thus, the PFPS-lists of items $\{a\}$ and $\{e\}$ are added to *boundPFPS*. For example, the PFPS-list of itemset $\{a\}$ is added to *boundPFPS* since $boundRa(\{a\}) = 1 \ge minRa$. Then, *boundPFPS* is sorted by the order $\succ$ of ascending *boundRa* values. Thereafter, the *Search* procedure is called to find larger PFPS (Algorithm 5).

The *Search* procedure performs a depth-first search by extending itemsets having their PFPS-lists in *boundPFPS*. It combines pairs of PFPS-lists of extensions of $\emptyset$ to generate 2-itemset extensions. The itemsets $\{a\}$ and $\{e\}$ are first combined to generate the itemset $\{a, e\}$ because they have the same prefix (the empty set). The *Intersect* procedure is applied on the PFPS-lists of $\{a\}$ and $\{e\}$ to generate the PFPS-list of $\{a, e\}$. The *sid-list* of $\{a\}$ is {0, 1, 2, 3} and that of $\{e\}$ is {0, 1, 3}. Thus, the *sid-list* of $\{a, e\}$ is {0, 1, 3}. For the first sequence, $\{a\}$'s tidlist is {0, 1, 2, 3, 4}, $\{e\}$'s tidlist is {0, 1, 3}, and their intersection is {0, 1, 3}. Thus, for the first sequence, the sequence identifier 0 is added to the *sid-list* of $\{a, e\}$ and the intersection {0, 1, 3} is added to the *tidlist-list* of $\{a, e\}$. The rest of the PFPS-list of $\{a, e\}$ is constructed in the same way for the other sequences. The PFPS-list of $\{a, e\}$ is shown in Table 6. Then, the *boundRa* value of $\{a, e\}$ is calculated using its PFPS-list. Since it is found that $boundRa(\{a, e\}) = 0.6 \ge minRa$, the itemset $\{a, e\}$ and its supersets may be PFPS. Hence, the sequences in which $\{a, e\}$ is periodic are found and $ra(\{a, e\})$ is calculated. Since $numSeq(\{a, e\}) = 3$, the ratio $ra(\{a, e\}) = 0.75 \ge minRa$, and thus $\{a, e\}$ is a PFPS and it is output. The PFPS-list of $\{a, e\}$ is added to *boundPFPS*. The same process is repeated for other itemsets, and the *Search* procedure is recursively called to explore the extensions of these itemsets.

4.4.2. Complexity

The complexity of the MPFPS$_{DFS}$ algorithm is analyzed as follows. The algorithm first scans the database to calculate $sup(\{i\}, s)$, $pr(\{i\}, s)$, $maxpr(\{i\}, s)$ and $stanDev(\{i\}, s)$ for each item $i \in I$. The time cost of this step is $\mathcal{O}(n \times w)$ where w is the average number of transactions per sequence. After that the algorithm performs a loop over each item to check if it is a PFPS and determine if the item should be added to *boundPFPS*. The time cost of this step is $\mathcal{O}(|I|)$. Items are then sorted in $\mathcal{O}(|I| \times \log(|I|))$ time. Then, the algorithm does a depth-first search by calling the *Search* procedure. A call to *Search* takes as parameters a set *ExtensionsOfP* containing PFPS-lists of k-itemsets having a same $(k-1)$-itemset as prefix. Then, the algorithm combines each pair of k-itemsets *Px* and *Py* to obtain an itemset *Pxy*. If *ExtensionsOfP* contains l itemsets, then $(l \times (l-1))/2$ pairs of itemsets are combined to generate $(k+1)$-itemsets. For each pair of itemsets *Px* and *Py*, the *Intersect* procedure is called. As previously explained for the MPFPS$_{BFS}$ algorithm, the time cost of this procedure is in the worst case $\mathcal{O}(n \times w)$. Then, the PFPS-list of *Pxy* is scanned to calculate *boundRa*(*Pxy*) and *ra*(*Pxy*), which has a time cost of at most $\mathcal{O}(n \times w)$. The algorithm explores the search space of itemsets using the above process. In the worst case, $2^{|I|} - 1$ itemsets are generated. Depending on how the algorithm's parameters are set, the pruning properties can more or less reduce the search space. The overall time cost of the MPFPS$_{DFS}$ algorithm is $\mathcal{O}(n \times w + |I| + |I| \times \log(|I|))$ to evaluate the 1-itemsets, and then $\mathcal{O}(n \times w)$ for each itemset considered in the search space.

In terms of space cost, the algorithm creates a PFPS-list for each itemset. During the depth-first search, at any moment, the algorithm only needs to keep the PFPS-lists of k-itemsets $(k \ge 1)$ sharing a same prefix. This thus generally requires much less memory than the MPFPS$_{BFS}$ algorithm, which requires in the worst to keep in memory all k- and $(k+1)$-itemsets. In the worst case, a PFPS-list takes $\mathcal{O}(n)$ space to store the $sid - list$ and $\mathcal{O}(n \times w)$ to store $tidlist - list$. Hence, the overall space cost of MPFPS$_{DFS}$ is $\mathcal{O}(n + n \times w)$ for each itemset.

Table 7
Database characteristics.

Database	# Items per trans.	# Distinct items	Avg. seq. length	# Sequences
FIFA	1	2990	34.74	20,450
Bible	1	13,905	21.6	36,369
T15I1KD300K	10	1000	15	30,000
Leviathan	1	9025	33.8	5,834

Table 8
Memory used by MPFPS$_{BFS}$ for different parameter values on *T15I1KD300K*.

maxStd Memory(MB) MPFPS$_{BFS}$(x,y)	3	4	5	10	15	20	1000
MPFPS$_{BFS}$(0,1000)	1938	2097	2086	2040	2,003	2,094	2,167
MPFPS$_{BFS}$(0,20)	1624	1663	1609	1646	1759	1845	1763
MPFPS$_{BFS}$(0.0001,20)	1673	1673	1575	1643	1664	1641	1607
MPFPS$_{BFS}$(0,10)	1,264	1,394	1,258	1457	1,457	1,460	1,492
MPFPS$_{BFS}$(0.0001,10)	1,234	1,384	1,230	1372	1,438	1,380	1,386

Table 9
Memory used by MPFPS$_{BFS}$ for different parameter values on *FIFA*.

maxStd Memory(MB) MPFPS$_{BFS}$(x,y)	1	2	3	4	5	10	1000
MPFPS$_{BFS}$(0,1000)	206	206	206	206	194	206	196
MPFPS$_{BFS}$(0,10)	246	251	251	246	251	251	251
MPFPS$_{BFS}$(0.0001,10)	225	246	246	225	246	225	246
MPFPS$_{BFS}$(0,5)	246	246	246	225	246	246	246
MPFPS$_{BFS}$(0.0001,5)	246	246	246	246	246	246	246

4.5. Correctness and completeness of the proposed algorithms

It can be easily found that the proposed MPFPS$_{BFS}$ and MPFPS$_{DFS}$ algorithms are both correct and complete. This is explained as follows. Since the depth-first search and breadth-first search procedures start from single items, and recursively append items to patterns to generate larger itemsets, the whole search space of all possible itemsets can be explored. To show that the algorithm is complete, it is necessary to show that no patterns are missed. Since the proposed algorithms only eliminate patterns using pruning strategies based on Theorem 5 and 6, which have been proved to only eliminate uninteresting patterns, no patterns are missed and the algorithms are complete. Besides, the algorithms are correct because the proposed PFPS-list data structure is sufficient to correctly calculate all measures.

5. Experimental evaluation

In prior work, a single algorithm named PHUSPM was proposed to mine periodic patterns in a sequence database. However, it finds patterns that regularly appear in a database whereas the proposed algorithms finds patterns that are periodic in many sequences, which are two very different problems. Thus, the performance of PHUSPM cannot be compared with that of the proposed algorithms. For this reason, this experimental evaluation compares the performance of the proposed MPFPS$_{BFS}$ and MPFPS$_{DFS}$ algorithms for several parameter values with baseline versions designed to find all frequent patterns. MPFPS$_{BFS}$ and MPFPS$_{DFS}$ are implemented in Java, and were run on a Windows 10 computer equipped with a 3.60 GHz Xeon E3 CPU with 64 GB of memory. The experiment is carried out on three real databases (*FIFA, Bible* and *Leviathan*) obtained from the website of the SPMF library [14], and on a synthetic sequence database created using the SPMF generator. These databases were chosen because they have different characteristics. Table 7 summarizes their characteristics. In *FIFA, Bible* and *Leviathan*, each transaction contains one item per transaction and all of these three are sparse databases with short transactions. For this reason, the synthetic *T15I1KD300K* database was also used, where each transaction has 10 items per transactions. It is a dense database with long transactions. The source code of the proposed algorithms and datasets can be obtained from http://www.philippe-fournier-viger.com/mpfps/MPFPS.rar.

The proposed algorithms have four parameters: *maxStd, minRa, maxPr* and *minSup*. The latter is used to find frequent patterns in each sequence and has been found to have little influence on the number of periodic patterns. Hence, *minSup* is set to a fixed value in the experiments ($minsup = 2$). Moreover, when *maxStd* and *maxPr* are set to large values and $minRa = 0$, the algorithm finds all frequent patterns. Thus, in the following, the MPFPS algorithm with $maxStd = 1000$, $minRa = 0$ and $maxPr = 1000$ will be used as baseline. The baseline will be compared with other parameter settings to assess the influence of the *maxStd, maxPr* and *minRa* parameters on the performance of the algorithm and number of patterns found. In the

Table 10
Memory used by MPFPS_{BFS} for different parameter values on *Bible*.

maxStd							
Memory(MB)							
MPFPS_{BFS}(x,y)	1	2	3	4	5	10	1000
MPFPS_{BFS}(0,1000)	317	264	322	322	343	322	343
MPFPS_{BFS}(0,10)	213	288	266	310	310	310	308
MPFPS_{BFS}(0.0001,10)	215	215	255	245	255	245	255
MPFPS_{BFS}(0,5)	240	245	225	245	245	255	245
MPFPS_{BFS}(0.0001,5)	251	251	255	255	250	250	251

Table 11
Memory used by MPFPS_{BFS} for different parameter values on *Leviathan*.

maxStd							
Memory(MB)							
MPFPS_{BFS}(x,y)	1	2	3	4	5	10	1000
MPFPS_{BFS}(0,1000)	201	201	201	201	201	202	207
MPFPS_{BFS}(0,20)	195	195	189	195	169	195	195
MPFPS_{BFS}(0.0001,20)	195	195	195	195	169	189	195
MPFPS_{BFS}(0,10)	87	87	87	87	87	87	87
MPFPS_{BFS}(0.0001,10)	87	87	87	87	87	87	87

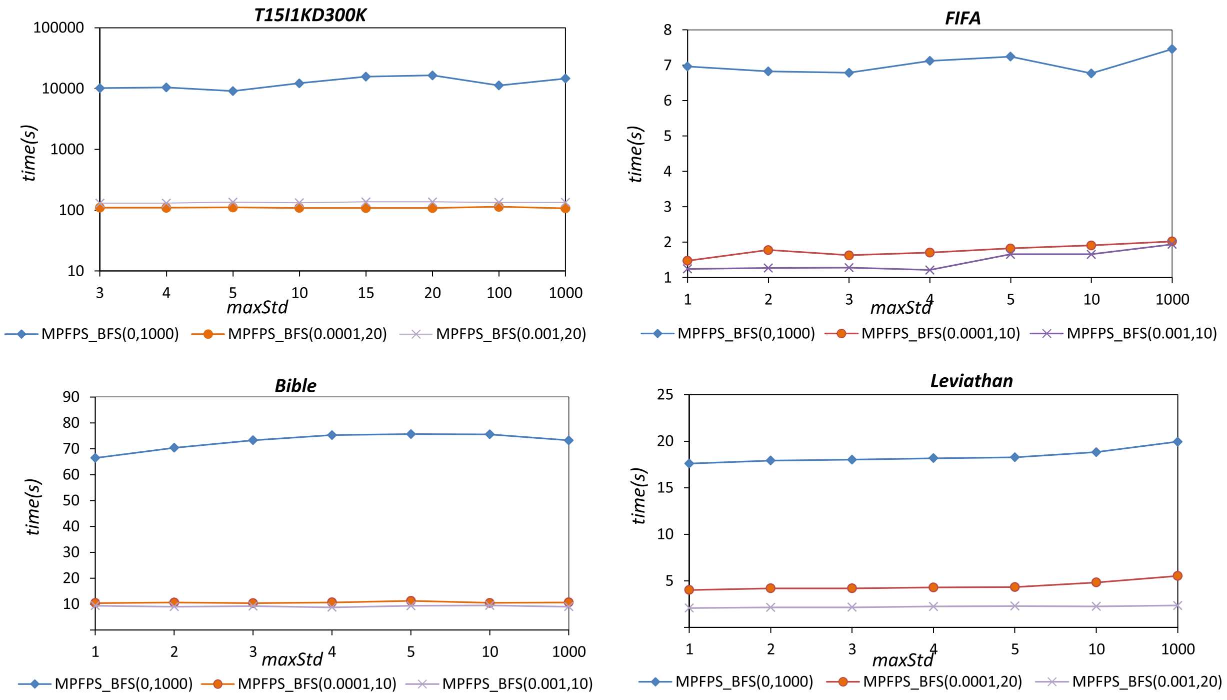


Fig. 3. Runtimes for various *minRa* and *maxStd* values for MPFPS_{BFS}.

following, *MPFPS*(*x*, *y*) denotes MPFPS with $minRa = x$, $maxPr = y$ and $minSup = 2$. Figs. 3–7 show the runtimes for various *maxStd*, *minRa* and *maxPr* values and Figs. 8 and show the number of patterns found from the four databases for different parameters values.

The *maxStd*, *minRa* and *maxPr* parameters are set according to the following considerations. First, to show the influence of *maxStd*, it must be set to small values. If was found that for the real and synthetic databases, values greater than 10 or 20 respectively do not influence much the performance. Hence, in the experiments for evaluating the influence of *maxStd*, it is set to values in the [1,5] and [3,15] intervals for the real and synthetic databases, respectively. In other experiments, it is set to 1000 to avoid considering its influence. Second, the *minRa* parameter was set to values that are small enough to find patterns. Since the databases are relatively sparse, appropriate values were empirically found to be 0.001 and 0.0001, respectively. Third, various values of the *maxPr* parameters were tested to evaluate its influence on the algorithms' performance. Generally, larger values will allow to prune less patterns and increase runtimes. It was found that values greater

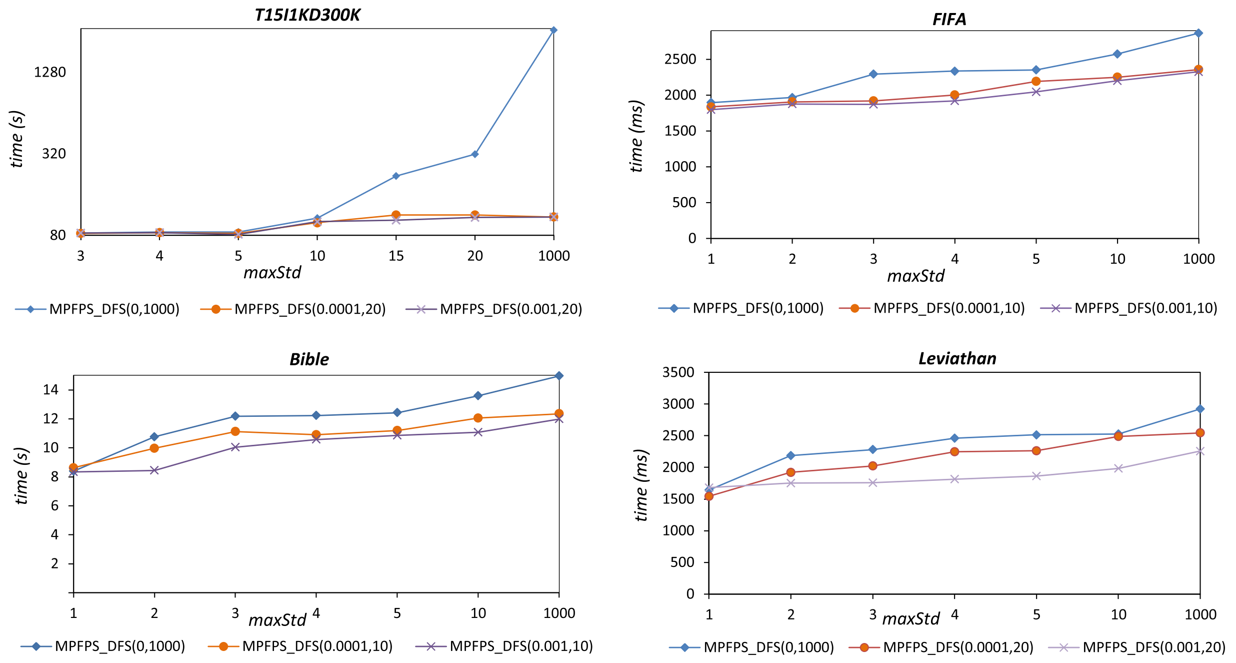


Fig. 4. Runtimes for various *minRa* and *maxStd* values for MPFPS$_{DFS}$.

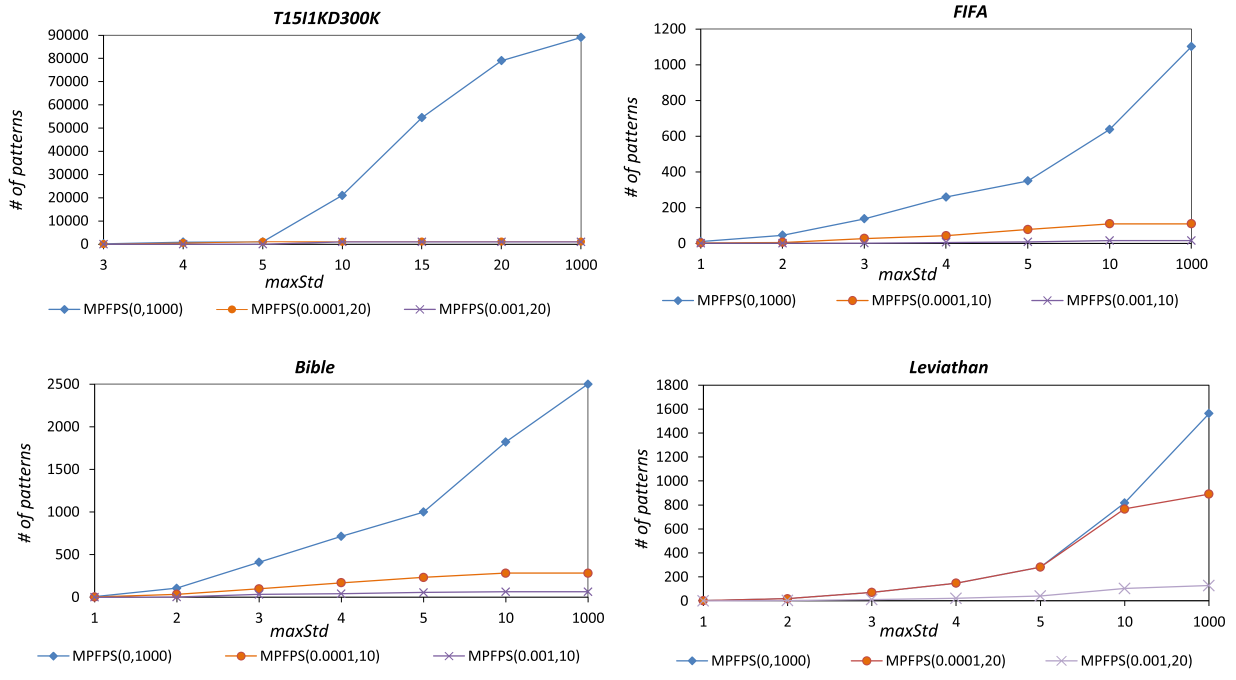


Fig. 5. Number of patterns for various *minRa* and *maxStd* values.

than 1000 did not influence much the performance. Thus, values of 5, 10, 20, and 1000 were used in the experiments to give an overview of the performance for different parameter values in the [1,1000] range.

5.1. *Influence of* minRa *and* maxStd

The *minRa* and *maxStd* parameters were first varied to evaluate their joint influence, while the *maxPr* parameter was set to a fixed value ($maxPr = 10$ for *FIFA* and *Bible*, and $maxPr = 20$ for *T15I1KD300K* and *Leviathan*). Figs. 3 and 4 show the

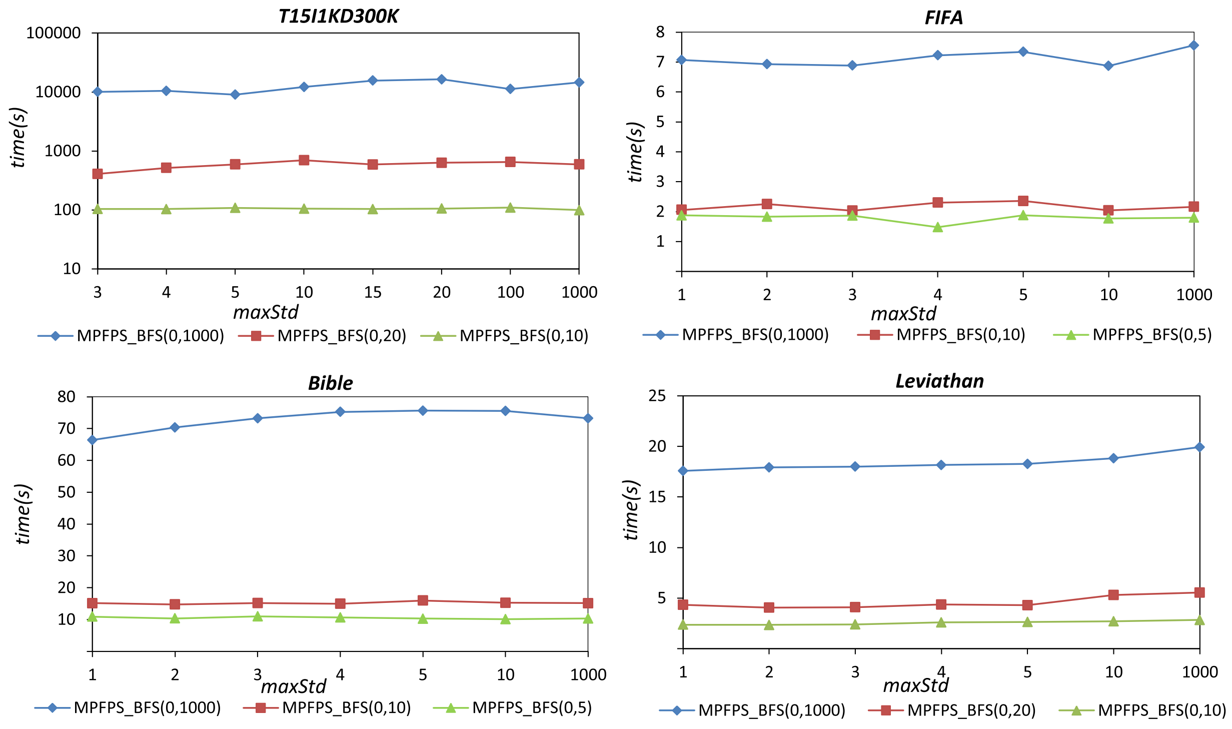


Fig. 6. Runtimes for various *maxStd* and *maxPr* values for MPFPS_{BFS}.

runtimes of the proposed MPFPS_{BFS} and MPFPS_{DFS} algorithms and Fig. 5 shows the number of PFPS found for various values of the *maxStd* and *minRa* parameters.

In Fig. 3, it can firstly be observed that while running the MPFPS_{BFS} algorithm, when the value of *maxStd* is increased, the runtime doesn't change much. For example, for $minRa = 0.0001$ and $maxPr = 20$ on *T15I1KD300K* and $minRa = 0.0001$ and $maxPr = 10$ on the other three databases, all the four lines are generally flat. This means that the *maxStd* parameter has little influence on the runtime of the MPFPS_{BFS} algorithm. Moreover, for all of the four databases, MPFPS_{BFS} was run with different *minRa* values to evaluate the influence of *minRa* on MPFPS_{BFS}. The figure shows that when $minRa = 0.0001$ or $minRa = 0.001$, the runtimes are a few times less than the baselines for the three real databases and nearly 100 times smaller for the synthetic database. This indicates the high practical value of the *minRa* parameter in reducing the search space for the MPFPS_{BFS} algorithm.

Fig. 4 shows the runtimes of the proposed MPFPS_{DFS} algorithms for various values of the *maxStd* and *minRa* parameters. In general, it is observed that mining PFPS in the three real databases *FIFA*, *Bible* and *Leviathan* can be up to 20% faster than mining all frequent patterns using the baseline algorithm, whereas for *T15I1KD300K*, it can be 30 times faster. The reason for this performance difference among databases is that there is only one item per transaction in *FIFA*, *Bible* and *Leviathan*, and thus the search space of itemsets is small, whereas transactions of *T15I1KD300K* contains many items per transactions, which results in a very large search space. Because a large number of frequent patterns in *T15I1KD300K* are non periodic, saving these patterns requires a lot space, and pruning non periodic patterns leads to a massive performance improvement. For the three real databases, this effect is much more limited.

It is observed in the Fig. 5 that, for the *maxStd* parameter, the number of periodic patterns can become very small compared to the baseline algorithm when this parameter is varied. For example, in *T15I1KD300K* there are 89,081 frequent patterns but only 20,986 PFPS when *maxStd* is decreased from 1000 to 10, and only 1003 PFPS when *maxStd* is decreased from 10 to 5. Also, it is found that the *minRa* parameter greatly influences the number of PFPS. For the *Bible* database and $maxStd = 1,000$, when *minRa* is varied from 0 to 0.001 (which means there should be at least 0.1% sequences in which an itemset is periodic), the number of PFPS decreases from 1126 to 63. This shows that most patterns are periodic in few sequences, and that the proposed algorithms can filter many non periodic patterns.

5.2. *Influence of* maxPr

The *maxPr* parameter was also varied to evaluate its influence. Since the average number of transactions per sequence in the datasets is not greater than 35, this parameter was set to small values. Moreover, $maxPr = 1,000$ was used as baseline, indicating that the parameter is deactivated. Figs. 6 and 7 compares the runtimes and Fig. 8 shows the number of PFPS found for various values of *maxPr*, when *maxStd* is varied as in the previous subsection.

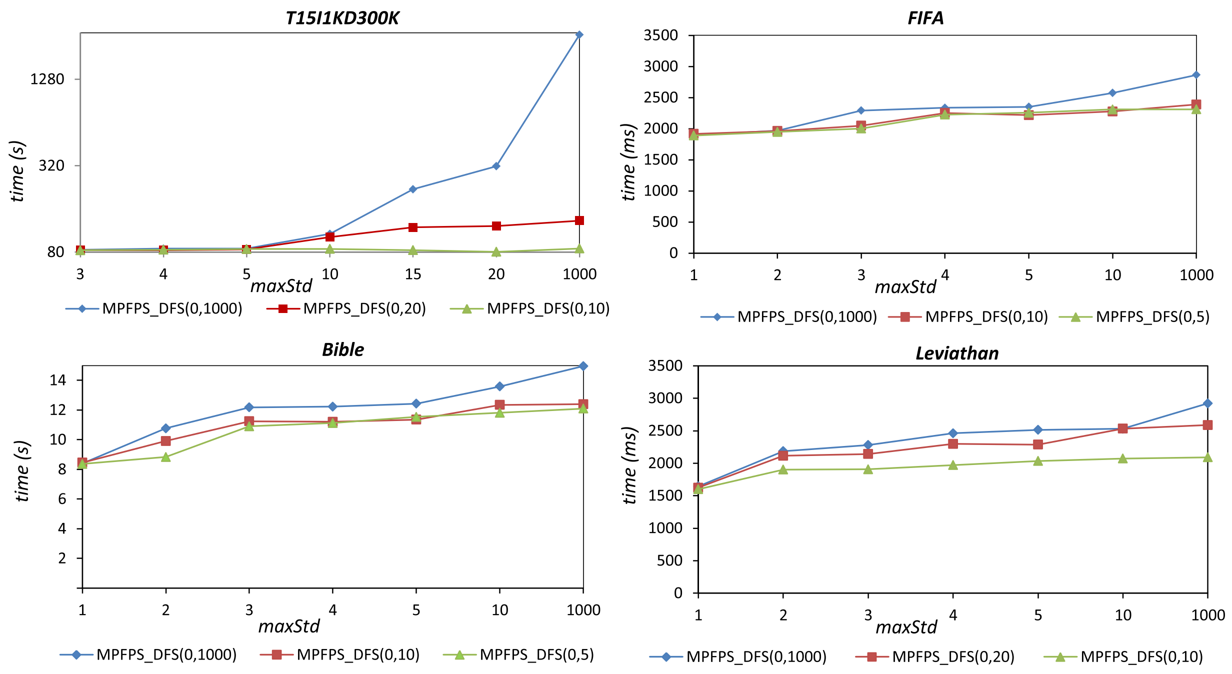


Fig. 7. Runtimes for various *maxStd* and *maxPr* values for MPFPS_{DFS}.

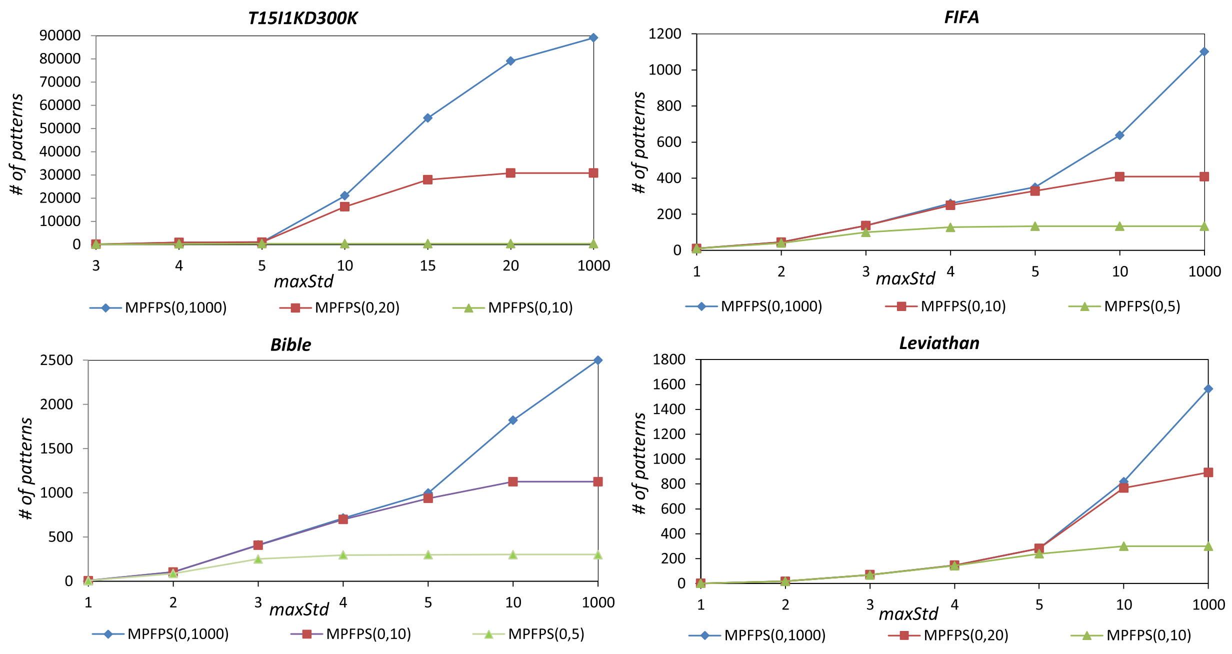


Fig. 8. Number of patterns for various *maxStd* and *maxPr* values.

It is first observed that the *maxPr* parameter can be used to greatly reduce the number of patterns found in Fig. 8. For example, on the *FIFA* dataset for $maxStd = 1,000$, $minRa = 0$ and $maxPr = 5,134$ patterns are found, while 1000 patterns are found for $maxPr = 1,000$. This is reasonable since the condition that the periods of a pattern must all be less than *maxPr* is very strict.

In terms of runtime, it can be seen in Fig. 6 that, decreasing the *maxPr* parameter of the MPFPS_{BFS} algorithm improves performance by a few times for the three real databases and nearly 100 times for the synthetic *T15KI1KD300K* database.

Similarly, it can be observed in Fig. 7 that for the MPFPS_{DFS} algorithm, decreasing the *maxPr* parameter can also improve performance but not by a large amount on the real *FIFA*, *Bible* and *Leviathan* datasets, while it provides a considerable im-

Table 12
Memory used by MPFPS$_{DFS}$ for different parameter values on *T15I1KD300K*.

maxStd Memory(MB) MPFPS$_{DFS}$(x,y)	3	4	5	10	15	20	1000
MPFPS$_{DFS}$(0,1000)	1837	1995	2035	2452	2,668	2,613	3,309
MPFPS$_{DFS}$(0,20)	1837	1995	2095	2272	2549	2,604	2773
MPFPS$_{DFS}$(0.0001,20)	1797	1876	2035	2233	2550	2550	2589
MPFPS$_{DFS}$(0,10)	1797	1837	1837	1837	1837	1837	1837
MPFPS$_{DFS}$(0.0001,10)	1797	1797	1797	1797	1797	1797	1797

Table 13
Memory used by MPFPS$_{DFS}$ for different parameter values on *FIFA*.

maxStd Memory(MB) MPFPS$_{DFS}$(x,y)	1	2	3	4	5	10	1000
MPFPS$_{DFS}$(0,1000)	280	320	320	320	320	400	525
MPFPS$_{DFS}$(0,10)	280	320	320	320	360	360	360
MPFPS$_{DFS}$(0.0001,10)	320	320	320	320	320	320	320
MPFPS$_{DFS}$(0,5)	280	320	320	320	320	320	320
MPFPS$_{DFS}$(0.0001,5)	320	280	320	280	320	280	280

provement on the synthetic dataset. This performance difference is caused by the number of internal candidates generated for these four databases. Since the MPFPS$_{BFS}$ algorithm generates a large amount of internal candidates on the synthetic database and *maxPr* is a very strict constraint, applying the pruning properties make an enormous performance improvement for this dataset.

5.3. Runtime comparison of MPFPS$_{\mathrm{BFS}}$ and MPFPS$_{\mathrm{DFS}}$

By comparing Fig. 3 with Fig. 4, it is found that for the same *minRa* and *maxStd* values, the performance difference between the baselines of MPFPS$_{BFS}$ and MPFPS$_{DFS}$ varies greatly. For the three real databases, the runtime of MPFPS$_{DFS}$ is several times less than the runtime of MPFPS$_{BFS}$: nearly 3 times in *FIFA*, 5 times in *Bible* and 7 times for *Leviathan*. But on the synthetic database, mining PFPS using MPFPS$_{DFS}$ is hundreds to thousands of times faster than using MPFPS$_{BFS}$ when the *maxStd* threshold is set to 20 or lower. This shows that MPFPS$_{DFS}$ is efficient for mining PFPS, using a depth-first search strategy, for the same *maxStd* and *minRa* values.

By comparing Fig. 6 with Fig. 7, the difference between the performance of the two proposed algorithms for the same *maxPr* values on the four databases can be observed. It is found that *maxPr* only influences the performance of MPFPS$_{DFS}$ to a small degree, while it can provide a considerable performance improvment for algorithm MPFPS$_{BFS}$, in the three real databases. In the synthetic database however, *maxPr* parameter can also make the runtimes for the MPFPS$_{DFS}$ algorithm. For the synthetic database and same *maxPr* values, the runtimes of MPFPS$_{DFS}$ are much less than the runtimes of MPFPS$_{BFS}$. For example, when *maxPr* is set to 20 and the other three parameters are set to $maxStd = 20$, $minRa = 0$ and $minSup = 2$, the runtime of MPFPS$_{DFS}$ is 121,679 while the runtime of MPFPS$_{BFS}$ is 631,940.

In summary, the MPFPS$_{DFS}$ algorithm generally outperforms the MPFPS$_{BFS}$algorithm in the experiments where the *maxStd*, *minRa* or *maxPr* parameters were varied. This is especially the case when these parameters are set to smaller, larger and smaller values, respectively, that is when the constraints on patterns to be found are more strict. Thus, if speed is important, the MPFPS$_{DFS}$ algorithm should be preferred.

Based on the above observations, and on the comparisons of the proposed algorithm with the baselines, it can be considered that the proposed algorithms are efficient in terms of runtime.

5.4. Memory used for different parameter values

Tables 12, 13, 14 and 15 compare the memory used for various values of *minRa* and *maxPr*, when *maxStd* is varied as in the previous subsection for the four databases.

It can firstly be observed that when *maxStd* is increased, both MPFPS$_{BFS}$ and MPFPS$_{DFS}$ need more space to store the PFPS-lists of the itemsets. For example, when *maxStd* is increased from 3 to 15 for MPFPS$_{DFS}$(0.0001, 20) in Table 12, memory usage also increases from 1,797 MB to 2,550 MB. This shows that *maxStd* can be used to greatly reduce the number of patterns stored in memory.

For the *minRa* parameter, it is observed that this parameter also influences memory usage. For example, consider MPFPS$_{DFS}$, for *T15I1KD300K* when *minRa* is increased from 0 to 0.0001, $maxStd = 4$ and $maxPr = 20$, the memory decreases from 1,995 MB to 1,876 MB. This shows that *minRa* is also helpful for reducing the number of stored patterns.

Table 14
Memory used by MPFPS$_{DFS}$ for different parameter values on *Bible*.

maxStd Memory(MB) MPFPS$_{DFS}$(x,y)	1	2	3	4	5	10	1000
MPFPS$_{DFS}$(0,1000)	280	320	380	400	520	847	1386
MPFPS$_{DFS}$(0,10)	280	320	360	400	480	560	560
MPFPS$_{DFS}$(0.0001,10)	280	320	320	320	320	320	320
MPFPS$_{DFS}$(0,5)	280	280	320	320	320	320	320
MPFPS$_{DFS}$(0.0001,5)	280	320	320	320	320	320	320

Table 15
Memory used by MPFPS$_{DFS}$ for different parameter values on *Leviathan*.

maxStd Memory(MB) MPFPS$_{DFS}$(x,y)	1	2	3	4	5	10	1000
MPFPS$_{DFS}$(0,1000)	67	72	72	77	87	201	330
MPFPS$_{DFS}$(0,20)	67	72	72	77	87	184	203
MPFPS$_{DFS}$(0.0001,20)	72	72	72	77	87	184	200
MPFPS$_{DFS}$(0,10)	67	67	72	77	82	82	82
MPFPS$_{DFS}$(0.0001,10)	67	67	67	72	72	77	77

The *maxPr* threshold can also reduce memory usage, as observed in the above tables. For example, consider MPFPS$_{DFS}$, for $T15I1KD300K$, when *maxPr* is decreased from 1000 to 10, $maxStd = 10$ and $minRa = 0$, the memory decreases from 2,452 MB to 1,837 MB.

Based on these tables, it is observed that, MPFPS$_{BFS}$ consumes less memory than MPFPS$_{DFS}$. For example, for $T15I1KD300K$, when $maxPr = 20$, $maxStd = 1000$ and $minRa = 0.0001$, 1,607 MB of memory is used by the MPFPS$_{BFS}$ algorithm and 2,589 MB of memory is used by the MPFPS$_{DFS}$ algorithm. And when *maxPr* is decreased to 5, 1,575 MB of memory is used by MPFPS$_{BFS}$ and 2,035 MB of memory is used by MPFPS$_{DFS}$. Thus, if memory usage is more important than runtime, the MPFPS$_{BFS}$ algorithm should be preferred.

5.5. Discussion

On overall, it has been found that the *maxPr, minRa* and *maxPr* parameters can be used to filter a large number of non periodic patterns. Thus, the proposed algorithms can be used to find a small set of periodic patterns. It was also shown that parameters can also reduce the runtime and the memory used. How to set the parameters is dataset dependent as different databases may contain patterns with shorter or longer periods, or periods that vary more or less greatly. It has been found that the *minSup* parameter has a very small influence on the algorithms' output and performance. Thus, it is recommended to set the *minSup* parameter to a small value and to increase this parameter only if performance is an issue. It is also recommended to set the *maxPr* parameter to a large value as the pruning condition for this parameter is very strict (as explained in the related work section). It should thus be used to filter patterns that have very large periods. Thus, the two parameters that should be considered as more important are the *maxStd* and *minRa* parameters. The former allows to specify the maximum variation in terms of periodicity over time for a periodic pattern. The latter is the ratio of sequences where a pattern must be periodic, and was introduced to be able to find patterns that are periodic in multiple sequences.

The patterns that were found by the proposed algorithms were also analyzed. Some interesting patterns have been found. For example, in the *Bible* dataset, for $maxStd = 2$ and $minRa = 0.001$, the two algorithms output three PFPS. The mined patterns are the words "the", "and" and "of", respectively. The word "the" is periodic frequent in 837 of the 36,369 sequences. Consider the 77th sequence as an example. The word "the" appears in the following transactions: {5, 10, 14, 20, 25, 32, 38, 41, 48, 55, 58, 61, 66}. The standard deviation is approximately 1.50, which is a very low value. This means the periods of the word "the" in this sequence is very stable. We have also performed an analysis of the patterns in the *Leviathan* text, which is a political work, also for $maxStd = 2$ and $minRa = 0.001$. The three mined periodic patterns are "of", "that" and "can". Although these two datasets share a common pattern "of", different periodic patterns are found in these two datasets. This is in accordance with studies on authorship attribution that have found that different authors will exhibit different patterns either in terms of the choices of words or the structure of sentences. There patterns thus represent the different writing styles of authors. In some related work, sequential patterns of different texts or ngrams have been used as features to design authorship attribution systems [39]. It would be interesting in future work to similarly evaluate the usefulness of periodic frequent patterns as feature for authorship attribution.

Patterns found in the *FIFA* dataset were also analyzed. This dataset contains sequences of click stream data from the website of the FIFA World Cup 98. For the same parameters values as above, it is found that webpage_17 is periodically clicked by 185 users. It shows the importance of this webpage during the world cup, as several persons are periodically checking this webpage over and over again to obtain updated information about the world cup.

The proposed algorithms have four parameters. For real life applications, it is recommended to set *maxPr* to a very large value to eliminate patterns having at least one very large period. The *minSup* and *minRa* parameters are used to make sure that the mined patterns are frequent in the whole database. Since the frequency distributions of items are different in each database, these parameters must be empirically set based on a database's characteristics. For dense databases, these parameters can be set to larger values, while for sparse databases, these parameters can be set to smaller values. As for the *maxStd* parameter, it should be set according to the user's preferences to find patterns that have a more or less stable periodic behavior. In future work, it would be interesting to design a method to automatically set the *minSup, minRa* and *maxPr* parameters as a function of each item's occurrences.

6. Conclusion

Previous work on periodic pattern mining have mainly focused on analyzing patterns in a single sequence. This paper has thus proposed a novel problem of mining periodic frequent patterns common to multiple sequences, which consists of discovering all patterns respecting user-defined minimum support, maximum periodicity, maximum standard deviation and minimum sequence periodic ratio thresholds. The problem was formally defined and properties of the problem have been studied. Moreover, two algorithms named MPFPS_{BFS} and MPFPS_{DFS} were proposed to discover these patterns, based on a novel PFPS-list structure and *boundRa* upper-bound to reduce the search space. Experiments on several databases have shown that the proposed algorithms are effective and can greatly reduce the number of patterns found by filtering non periodic patterns.

There are several opportunity for future work. First, this paper did not make any assumptions on whether periods of each itemset in sequences follows some statistical distribution. In future work, we will consider adapting the proposed model for various distributions. Furthermore, we will consider extending the model to discover more complex types of periodic patterns such as partial orders, rules and sequential patterns in sequences, and applications to classification [47].

Acknowledgments

This work was supported by funding from the National Science Foundation of China.

References

[1] R. Agrawal, R. Srikant, Fast algorithms for mining association rules, in: Proceedings of the Twentieth International Conference on Very Large Data Bases, Morgan Kaufmann, Santiago de Chile, 1994, pp. 487–499.
[2] R. Agrawal, R. Srikant, Mining sequential patterns, in: Proceedings of the Eleventh International Conference on Data Engineering, IEEE Press, Taipei, 1995, pp. 3–14.
[3] K. Amphawan, A. Surarerks, P. Lenca, Mining periodic-frequent itemsets with approximate periodicity using interval transaction-ids list tree, in: Proceedings of the Third International Conference on Knowledge Discovery and Data Mining, 2010, pp. 245–248. Phuket.
[4] K. Amphawan, P. Lenca, A. Surarerks, Mining top-k periodic-frequent pattern from transactional databases without support threshold, in: Proceedings of the First International Conference on Advances in Information Technology, Springer, Bangkok, 2009, pp. 18–29.
[5] J. Ayres, J. Flannick, J. Gehrke, T. Yiu, Sequential pattern mining using a bitmap representation, in: Proceedings of the Eighth ACM SIGKDD International Conference Knowledge Discovery and Data Mining, ACM, Edmonton, 2012, pp. 429–435.
[6] M. Berlingerio, F. Pinelli, F. Calabrese, ABACUS: frequent pattern mining-based community discovery in multidimensional networks, J. Data Min. Knowl. Discov. 26 (2013) 294–320.
[7] H. Cao, D.W. Cheung, N. Mamoulis, Discovering partial periodic patterns in discrete data sequences, in: Proceedings of the Eighth Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, Sydney, 2004, pp. 653–658.
[8] T. Dinh, V.N. Huynh, B. Lee, Mining periodic high utility sequential patterns, in: Proceedings of the Ninth Asian Conference on Intelligent Information and Database Systems, Springer, Kanazawa, 2017, pp. 545–555.
[9] Y. Duan, X. Fu, B. Luo, Z. Wang, J. Shi, X. Du, Detective: automatically identify and analyze malware processes in forensic scenarios via DLLs, in: Proceedings of the International Conference on Communications, IEEE, London, 2015, pp. 5691–5696.
[10] Y. Fan, Y. Ye, L. Chen, Malicious sequential pattern mining for automatic malware detection, J. Expert Syst. Appl. 52 (2016) 16–25.
[11] B. Fernando, F. Elisa, T. Tinne, Effective use of frequent itemset mining for image classification, in: Proceedings of the Twelfth European Conference on Computer Vision, Springer, Florence, 2012, p. 214227.
[12] M. Fidino, S.B. Magle, Using fourier series to estimate periodic patterns in dynamic occupancy models, Ecosphere 8 (9) (2017). E01944. doi: 10.1002/ecs2.1944.
[13] P. Fournier-Viger, A. Gomariz, M. Campos, R. Thomas, Fast vertical mining of sequential patterns using co-occurrence information, in: Proceedings of the Eighteenth Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, Tainan, 2014, pp. 40–52.
[14] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C. Wu., V.S. Tseng, SPMF: a java open-source pattern mining library, J. Mach. Learn. Res. 15 (1) (2014) 3389–3393.
[15] P. Fournier-Viger, T. Gueniche, S. Zida, V.S. Tseng, ERMiner: sequential rule mining using equivalence classes, in: Proceedings of the Thirteenth International Symposium on Intelligent Data Analysis, Springer, Leuven, 2014, pp. 108–119.
[16] P. Fournier-Viger, Z. Li, J.C.W. Lin, R.U. Kiran, H. Fujita, Discovering periodic patterns common to multiple sequences, in: Proceedings of the Twentieth International Conference on Big Data Analytics and Knowledge Discovery, 2018, pp. 231–246. Regensburg.
[17] P. Fournier-Viger, C.-W. Lin, Q.-H. Duong, T.-L. Dam, PHM: mining periodic high-utility itemsets, in: Proceedings of the Sixteenth Industrial Conference on Data Mining, Springer, New York, 2016, pp. 64–79.
[18] P. Fournier-Viger, C.-W. Lin, Q.-H. Duong, T.-L. Dam, L. Sevcic, D. Uhrin, M. Voznak, PFPM: discovering periodic frequent patterns with novel periodicity measures, in: Proceedings of the Second Czech-China Scientific Conference, 2017, pp. 1–13.
[19] P. Fournier-Viger, J.C.-W. Lin, R.U. Kiran, Y.S. Koh, R. Thomas, A survey of sequential pattern mining, J. Data Sci. Pattern Recognit. 1 (1) (2017) 54–77.
[20] P. Fournier-Viger, J.C.W. Lin, B. Vo, T.T. Chi, J. Zhang, H.B. Le, A survey of itemset mining, J. WIREs Data Min. Knowl. Discov. 7 (4) (2017) 18.
[21] P. Fournier-Viger, C.W. Wu, V.S. Tseng, L. Cao, R. Nkambou, Mining partially-ordered sequential rules common to multiple sequences, IEEE Trans. Knowl. Data Eng. 27 (8) (2015) 2203–2216.
[22] P. Fournier-Viger, R. Nkambou, E. Mephu Nguifo, A knowledge discovery framework for learning task models from user interactions in intelligent tutoring systems, in: Proceedings of the Seventh Mexican International Conference on Artificial Intelligence, Springer, Atizapn de Zaragoza, 2008, pp. 765–778.

[23] K. Gouda, M. Hassaan, M.J. Zaki, Prism: an effective approach for frequent sequence mining via prime-block encoding, J. Comput. Syst. Sci. 76 (1) (2010) 88–102.
[24] S. Halder, M. Samiullah, Y.-K. Lee, Supergraph based periodic pattern mining in dynamic social networks, J. Expert Syst. Appl. 72 (2017) 430–442.
[25] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M. Hsu, FreeSpan: frequent pattern-projected sequential pattern mining, in: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, NewYork, 2000, pp. 355–359.
[26] J. Han, J. Pei, Y. Yin, Mining frequent patterns without candidate generation, in: Proceedings of the ACM SIGMOD International Conference on Management of data, ACM, Dallas, 2000, pp. 1–12.
[27] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, M. Hsu, PrefixSpan: mining sequential patterns efficiently by prefix-projected pattern growth, in: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM, Boston, 2000, pp. 355–359.
[28] S.K. Harms, J. Deogun, T. Tadesse, Discovering sequential association rules with constraints and time lags in multiple sequences, in: Proceedings of the Thirteenth International Symposium Methodologies on Intelligent Systems, Springer, North-Holland, 2002, pp. 373–376.
[29] R.U. Kiran, M. Kitsuregawa, P.K. Reddy, Efficient discovery of periodic-frequent patterns in very large databases, J. Syst. Softw. 112 (2016) 110–121.
[30] R.U. Kiran, P.K. Reddy, Mining rare periodic-frequent patterns using multiple minimum supports, in: Proceedings of the Fifteenth International Conference on Management of Data, IEEE, Mysore, 2009, pp. 7–8.
[31] R.U. Kiran, P.K. Reddy, An alternative interestingness measure for mining periodic-frequent patterns, in: Proceedings of the Sixteenth International Conference on Database Systems for Advanced Applications, HongKong, HongKong, 2011, pp. 183–192.
[32] D. Li, J.S. Deogun, Discovering partial periodic sequential association rules with time lag in multiple sequences for prediction, in: Proceedings of the Eleventh International Symposium on Methodologies for Intelligent Systems, Springer, New York, 2005, pp. 332–341.
[33] J.C.-W. Lin, W. Gan, P. Fournier-Viger, H.-C. Chao, W.J. T., J. Zhan, Extracting recent weighted-based patterns from uncertain temporal databases, Eng. Appl. Artif. Intell. 61 (2017) 161–172.
[34] Y. Liu, Y. Zhao, L. Chen, J. Pei, J. Han, Mining frequent trajectory patterns for activity monitoring using radio frequency tag arrays, J. Trans. Parallel Distrib. Syst. 23 (11) (2012) 2138–2149.
[35] D. Lo, G. Ramalingam, V.P. Ranganath, K. Vaswani, Mining quantified temporal rules: formalism, algorithms, and evaluation, in: Proceedings of the Sixteenth Working Conference on Reverse Engineering, IEEE, Lille, 2009, pp. 62–71.
[36] H. Mannila, H. Toivonen, A.I. Verkamo, Discovery of frequent episodes in event sequences, J. Data Min. Knowl. Discov. 1 (3) (1997) 259–289.
[37] E. Mwamikazi, P. Fournier-Viger, C. Moghrabi, R. Baudouin, A dynamic questionnaire to further reduce questions in learning style assessment, in: Proceedings of the Tenth International Conference on Artificial Intelligence Applications and Innovations, Springer, Rhodes, 2014, pp. 224–235.
[38] M.A. Nishi, C.F. Ahmed, M. Samiullah, B. Jeong, Effective periodic pattern mining in time series databases, J. Expert Syst. Appl. 40 (2013) 3015–3027.
[39] J.M. Pokou, P. Fournier-Viger, C. Moghrabi, Authorship attribution using small sets of frequent part-of-speech skip-grams, in: Proceedings of the Twenty Ninth International Florida Artificial Intelligence Research Society Conference, AAAI, Key Largo, 2016, pp. 86–91.
[40] D. Schweizer, M. Zehnder, H. Wache, H.F. Witschel, D. Zanatta, M. Rodriguez, Using consumer behavior data to reduce energy consumption in smart homes: applying machine learning to save energy without lowering comfort of inhabitants, in: Proceedings of the Fourteenth International Conference on Machine Learning and Applications, IEEE, Miami, 2015, pp. 1123–1129.
[41] R. Srikant, R. Agrawal, Mining sequential patterns: generalizations and performance improvements, in: Proceedings of the EDBT Advances in Database Technology, 1996, pp. 1–17.
[42] A. Surana, R.U. Kiran, P.K. Reddy, An efficient approach to mine periodic-frequent patterns in transactional databases, in: Proceedings of the Sixteenth Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, Kuala Lumpur, 2011, pp. 254–266.
[43] T. Syed, A. Chowdhury, J. Byeong, C. Young L., Discovering periodic-frequent patterns in transactional databases, in: Proceedings of the Thirteenth Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, Bangkok, 2009, pp. 242–253.
[44] T. Uno, M. Kiyomi, H. Arimura, LCM ver. 2: efficient mining algorithms for frequent/closed/maximal itemsets, in: Proceedings of the Fourth International Conference on Data Mining Workshop on Frequent Itemset Mining Implementations, IEEE, CEUR, 2004.
[45] J. Wang, J. Han, C. Li, Frequent closed sequence mining without candidate maintenance, J. IEEE Trans. Knowl. Data Eng. 19 (8) (2007) 1042–1056.
[46] M.J. Zaki, K. Gouda, Fast vertical mining using diffsets, in: Proceedings of the Ninth ACM SIGKDD Intern. Conf. on Knowledge Discovery and Data Mining, ACM, Washington, 2003, pp. 326–335.
[47] C. Zhang, C. Liu, X. Zhang, G. Almpanidis, An up-to-date comparison of state-of-the-art classification algorithms, Expert Syst. Appl. 82 (2017) 128–150.
[48] S. Ziebarth, I.A. Chounta, H.U. Hoppe, Resource access patterns in exam preparation activities, in: Proceedings of the Fifth European Conference on Technology Enhanced Learning, Springer, Toledo, 2015, pp. 497–502.
[49] A. Zimmermann, Understanding episode mining techniques: benchmarking on diverse, realistic, artificial data, J. Intell. Data Anal. 18 (5) (2014) 761–791.