Data Mining and Knowledge Discovery (2021) 35:1225–1257
https://doi.org/10.1007/s10618-021-00753-9

Mining full, inner and tail periodic patterns with perfect, imperfect and asynchronous periodicity simultaneously

Jen-Wei Huang[1] · **Bijay Prasad Jaysawal**[1] · **Cheng-Chung Wang**[2]

Received: 25 August 2019 / Accepted: 12 March 2021 / Published online: 5 April 2021

Abstract

Periodic pattern has been utilized in many real life applications, such as weather conditions in a particular season, transactions in a superstore, power consumption, computer network fault analysis, and analysis of DNA and protein sequences. Periodic pattern mining is a popular though challenging research field in data mining because periodic patterns are of different types (namely full, inner, and tail patterns) and varied periodicities (namely perfect, imperfect, and asynchronous periodicity). Previous periodic pattern mining methods have some disadvantages: (1) Previous methods have to find different patterns separately; (2) They require postprocessing such as level-by-level join strategies for mining complex periodic patterns which have wildcards between two items. They cannot mine full, tail, and inner periodic patterns with perfect, imperfect, and asynchronous periodicities simultaneously. Therefore, an effective and comprehensive approach capable of discovering the above specified kinds of periodic patterns is needed. We propose a novel suffix tree-based algorithm, Mining dIfferent kinds of Periodic Patterns Simultaneously, MIPPS, to address the above issues. MIPPS finds different kinds of periodic patterns with different periodicities simultaneously without level-by-level join techniques using a novel incremental propagation generator. In addition, MIPPS mines periodic patterns efficiently using some pruning strategies. For the performance evaluation, we use both synthetic and real data to confirm good performance and scalability with complex periodic patterns.

Responsible editor: Eamonn Keogh.

✉ Jen-Wei Huang
jwhuang@mail.ncku.edu.tw

Bijay Prasad Jaysawal
bijay@jaysawal.com.np

Cheng-Chung Wang
smirkchung@gmail.com

[1] Department of Electrical Engineering, National Cheng Kung University, Tainan, Taiwan

[2] Yuan Ze University, Taoyuan City, Taiwan

Keywords Data mining · Periodic pattern · Periodicity · Symbolic sequence data

1 Introduction

A time series is a collection of data values gathered at uniform time intervals to reflect the behavior of an entity. Some examples of time series are the weather conditions of a particular place, transactions in a superstore, power consumption records, computer network monitoring logs. Time series data mining (Fu 2011; Esling and Agon 2012) has been applied in a wide range of real-life problems in various fields of research, such as economic forecasting (Song and Li 2008), intrusion detection (Zhong et al. 2007), medical surveillance (Burkom et al. 2007), gene expression analysis (ho Lin et al. 2008), hydrology (Ouyang et al. 2010), human behavior analysis (Pierson et al. 2018), and biological data (DNA and protein sequences) analysis (Katti et al. 2000; Ahdesmäki et al. 2005; Glynn et al. 2005).

Some time series can be characterized by being composed of recurrent cycles. For instance, power consumption has higher demand in summer, animals have certain migration route during a year and so on. Identifying repeating periodic patterns may reveal important observations about the behavior and future trends of the case represented by the time series (Weigend and Gershenfeld 1994), and hence leads to more effective decision making. In addition, periodicity detection is a process for finding temporal regularities within the time series. In other words, a goal of analyzing a time series is to determine whether and how frequent a periodic pattern is repeated within the time series (Rasheed et al. 2011). Mining periodic patterns (Berberidis et al. 2002; Elfeky et al. 2005a; He et al. 2008) in time series databases is a popular research field in data mining. Varied periodicities (Han et al. 1999; Yang et al. 2003; Ozden et al. 1998; Pujeri and Karthik 2012; Elfeky et al. 2005b; Sheng et al. 2006) have many different applications.

The time series data could be discretized into finite symbols taken from an alphabet set $\sum$. For example, SAX (Symbolic Aggregate approXimation) representation (Lin et al. 2007), discritization approach discussed in (Rasheed et al. 2011), et cetera, can be used to discretize a time series into sequence of symbols. In this study, we do not focus on discretization method. We assume that the time series data has been already discretized into sequence of symbols. In different applications, different discretization methods may be suitable which requires domain knowledge. In addition, there are some applications where data exist in the form of sequence of symbols, such as protein sequence. Let us take an example of discrete time series $T = aababc$ with length $n = 6$ and each value is taken from the alphabet set {a,b,c}. A pattern P is a non-empty sequential subset of a time series. When a pattern appears periodically in time series, it is known as a periodic pattern. For example, given a time series $T = abababacabccbcbb$, there is a periodic pattern ab and its period is 2 within time points t_1–t_6. In addition, the wildcard character, $*$, represents any item or it can be viewed as an unimportant item. There are two periodic patterns containing wildcard characters in the above time series T. The period of $aba*$ is 4 within time points t_1–t_8, and the period of $c * b$ is 3 within time points t_8–t_{16}.

Table 1 Periodic patterns with different periodicities and period 2

Time series	Pattern	Periodicity	Confidence
$\frac{ab}{\surd\surd}\ \frac{ab}{\surd\surd}\ \frac{ab}{\surd\surd}$	ab	Perfect periodicity	3/3
$\frac{ab}{\surd\surd}\ \frac{ab}{\surd\surd}\ \frac{ac}{\times\times}\ \frac{ab}{\surd\surd}\ \frac{cd}{\times\times}$	ab	Imperfect periodicity	3/5
$\frac{ab}{\surd\surd}\ \underline{c}\ \frac{ab}{\surd\surd}\ \underline{ac}\ \frac{ab}{\surd\surd}\ \frac{ab}{\surd\surd}$	ab	Asynchronous periodicity	4/5

The periodic patterns found by previous works can be classified as full and partial periodic patterns, perfect and imperfect periodic patterns, synchronous and asynchronous periodic patterns (Sirisha et al. 2014). Full periodic patterns are patterns where each position in the period contains an item from the set of items. For example, abc is a full periodic pattern of period 3. Partial periodic patterns (Han et al. 1999) are the patterns with wildcard character, $*$, which represents any item or unimportant items. For example, both $a * c$ and $ab*$ are partial periodic patterns of period 3. For details about different kinds of periodic patterns and related mining algorithms, we refer the readers to the survey paper (Sirisha et al. 2014).

Now, we describe the periodic patterns based on periodicities, i.e. perfect and imperfect, synchronous and asynchronous. In a given time series, from the first occurrence of the pattern to the end of the time series, if there is no gap between any two successive patterns, the periodicity is called perfect periodicity (Ozden et al. 1998). The patterns with perfect periodicity are called perfect periodic patterns. Table 1 shows examples of periodic patterns with different periodicities having period 2. For example, $T = ababab$ has the pattern ab with period 2 and perfect periodicity. Usually, the perfection of periodicity is represented by confidence. The confidence is the ratio of the actual frequency to the maximum possible frequency of the pattern in the time series. In the above example of perfect periodicity, the pattern has the actual frequency 3 and maximum possible frequency 3, so the confidence is 3/3. In perfect periodicity, the confidence of patterns is always 100% because all possible occurrences of patterns are present in the time series. It is possible to have some of the expected occurrences of periodic patterns missing. These periodic patterns with the absence of possible occurrences at some time points are said to have imperfect periodicity. The confidence of imperfect periodicity is always less than 100% because some occurrences of patterns are missing in the time series. The patterns with imperfect periodicities are called imperfect periodic patterns. For example, the time series $T = ababacabcd$ has a periodic pattern ab with period 2 and imperfect periodicity. In this example, the confidence of the pattern ab is 3/5. Furthermore, real data may contain noises or outliers that will disturb the periodicity, i.e., periodicity may not be synchronous. Asynchronous periodicity (Yang et al. 2003) allows a gap of different lengths to deal with noise between two successive patterns. The patterns with asynchronous periodicity are called asynchronous periodic patterns. The algorithm to find asynchronous periodicity in (Yang et al. 2003) defines two parameters min_rep and max_dis, a.k.a. gap. A valid segment of time series contains contiguous min_rep occurrences of patterns. If the gap between two successive valid segments are no more than max_dis, those segments are connected and the periodic patterns are said to have asynchronous

periodicity. In our work, for simplicity, we assume min_rep equals to 1 for a valid segment so that all possible patterns can be found and user does not need to set parameter for this. Note that this parameter can still be easily applied in our algorithm if it is required. For example, given a time series $T = abcabacabab$ and $gap = 2$, there is a periodic pattern ab with period 2 and asynchronous periodicity. In this example, the confidence of the ab pattern is 4/5.

Rasheed *et al.* in (Rasheed et al. 2011) proposed STNR algorithm to mine the symbol periodicity, sequence periodicity, and segment periodicity using suffix tree. Symbol and sequence periodicities are kind of partial periodic patterns whereas segment periodicity can be regarded as full periodic patterns. The authors also proposed to mine periodic patterns with time tolerance which is the variation of asynchronous periodic patterns. They also discussed to mine time series in a subsection of time series by defining start and end position. By considering subsection of the time series, the periodic pattern may have different periodicity. However, STNR cannot generate patterns like $ab*b$ that contains wildcard or unimportant event in between two events (Nishi et al. 2013; Sirisha et al. 2014). Nishi et al. (Nishi et al. 2013) observed this limitation of STNR and used apriori-like level-by-level pattern mining approach to generate the patterns containing wildcard in between two events such as $a * b$ and $ab * *d$.

Partial periodic pattern was first proposed in (Han et al. 1999). Partial periodic patterns allow wildcards which represent any item or unimportant item at the position of wildcard. In this study, we propose to further classify the partial periodic pattern into two types, one is tail periodic pattern and the other one is inner periodic pattern. For the tail periodic pattern, the last item of the pattern is a wildcard, e.g., $ab*$. For inner periodic pattern, the first and last item of the pattern cannot be a wildcard and the wildcards are located between any two items, e.g., $a * b$ and $a * *c$. Both types of periodic patterns have their individual peculiarities and we can use their specific features to devise mining strategy. Inner pattern did not receive much attention previously. However, we believe that inner pattern has significant importance and it is more complex to find inner patterns than tail patterns in periodic pattern mining. For example, during school days, a student leaves home, buys breakfast from a random store, and then goes to school regularly each day. There are three items in the periodic pattern and the middle item of the pattern is wildcard because the student may go to different stores to buy breakfast. This kind of behavior is common in real life and can be captured as an inner periodic pattern.

Han et al. (Han et al. 1999) and Nishi et al. (Nishi et al. 2013) used apriori-like level-by-level pattern mining approach to generate the complex inner pattern. Yang et. al (Yang et al. 2003) found complex inner patterns by joining shorter known frequent periodic patterns. Such methods generate unnecessary candidates and are time-consuming. In addition, it is unfavorable for pruning redundant periodic inner patterns. Therefore, how to find periodic inner patterns efficiently remains a difficult problem.

In this work, we devise a novel algorithm to find full, inner, and tail periodic patterns with perfect, imperfect and asynchronous periodicities simultaneously. We also consider finding the periodic patterns in the subsection of time series. Previous works (Nishi et al. 2013) and (Sirisha et al. 2014) observed the limitation of the STNR, which uses the suffix tree to generate the occurrence vectors of periodic patterns cannot

generate the inner patterns. In this work, we devise a novel suffix tree-based approach to generate the occurrence vectors of inner patterns along with other patterns. The proposed novel suffix tree-based algorithm named Mining dIfferent kinds of Periodic Patterns Simultaneously, MIPPS, mines all these full, tail and inner patterns with different periodicities (perfect, imperfect and asynchronous) simultaneously.

Moreover, different periods have different kinds of periodic patterns. For example, given a time series $T = ababcabd$, the pattern ab is a perfect periodic pattern when its period is 2 within time points t_1 to t_6. When period is 3 and gap is 1, there is a partial periodic pattern $ab*$ with asynchronous periodicity in the entire time series. In order to find all possible periodic patterns, we need to detect all possible periods in the time series data, which makes discovering all the periodic patterns challenging. Therefore, we further propose some optimization strategies for detecting periodic patterns and prune unnecessary patterns to make MIPPS more efficient.

In the experiments, we compare the performance of MIPPS algorithm with another suffix tree-based algorithm STNR (Rasheed et al. 2011). We use synthetic and real data in our experiments to investigate performances in different conditions such as the number of distinct items, the size of periods for periodic patterns and various lengths of the time series data. We also analyze memory usage for different steps of the algorithm, which are *building suffix tree* and *discovering periodic patterns from suffix tree*. The results of the experiments show that MIPPS outperforms STNR in mining full, inner, and tail periodic patterns from time series data. We also apply MIPPS on an important application of periodic pattern mining in biological data. According to Katti et al. (Katti et al. 2000), the protein sequence P17437 (Skin secretory protein xP2) has a known periodic pattern $\{APAPA**E**\}$ and there are 25 repeats of this pattern. MIPPS was able to determine the periodic pattern $\{APAPA**E**\}$ with 25 repeats, which demonstrates that MIPPS can be applied to real life data. Moreover, other interesting periodic patterns with repeats more than 25 were found by MIPPS.

The main contributions of our work can be summarized as follows:

- We propose a suffix tree-based data structure and a top down traversal approach to generate three types of patterns and their occurrence positions in a discrete time series (sequence of symbols) with only one scan.
- We propose a single method, MIPPS, to mine full, tail and inner patterns with different kinds of periodicities (perfect, imperfect, and asynchronous) simultaneously.
- A number of optimization strategies are presented to efficiently detect periodic patterns and prune unnecessary patterns and periods.
- MIPPS is shown to be applicable to real biological datasets such as DNA and protein sequences. In addition, MIPPS is also able to find some other novel periodic patterns.

The rest of the paper is organized as follows: Section 2 presents related works. Section 3 discusses the preliminaries of periodic patterns in time series. Section 4 describes the proposed algorithm along with the suffix tree-based representation, the utilized optimization and pruning strategies. Section 5 discusses the experimental results using both real and synthetic data. Finally, Sect. 6 concludes this study.

2 Related works

Periodic pattern mining emerged from association rules. The work, proposed by Ozden et al. (Ozden et al. 1998), observed that the confidence of some rules are very low when considering total length of time. But when we specify a length of time, the confidence is high. They developed a sequential algorithm to discover cyclic patterns. It uses cycle-skipping to reduce access times, cycle-pruning to avoid repeated search of some sub-patterns of known periodic patterns and the cycle-elimination to eliminate the infrequent patterns. Han et al. (Han et al. 1999) noticed a useful related type of periodic pattern, called partial periodic pattern. They proposed partial periodic pattern with wildcard which represents any item or unimportant item in the sequence. Their proposed max-subpattern hit-set starts from discovering frequent 1-patterns and then uses the frequent 1-pattern to generate max-pattern. Using the max-pattern hit-set, their algorithm finds the max subpatterns and builds a max-subpattern tree. The resulting tree can answer the number of occurrences of partial periodic patterns in the time series and helps to mine frequent partial periodic patterns. Their algorithm needs only two scans of the time series database. Cao et al. (Cao et al. 2004) presented a new data structure named Abbreviated List Table (ALT) to improve max-subpattern hit-set for discovering frequent 1-patterns over different periods. ALT can find the frequent 1-patterns of different periods in a single scan of the time series database.

Real life data may have noises and outliers that can disturb the synchronous behavior of periodic patterns. Yang et al. (Yang et al. 2003) defined the maximum allowed disturbance between two successive valid segments to resolve the noise problem in real life data. It is called asynchronous periodic pattern. Their algorithm can find the longest subsequence of asynchronous periodic pattern. Huang and Chang (Huang and Chang 2005) proposed a general model for mining asynchronous periodic patterns in temporal databases (SMCA). Temporal databases may have multiple items at one time point. They arrange data in a vertical format which is more efficient than a horizontal database. SMCA mines periodic segments starting from a single event and uses two methods to generate multi-event patterns by timelist-based enumeration and segment-based enumeration. The inner pattern is produced by combining valid segments of the same period in depth-first order.

Previous researches employ different techniques to transform the time series data into other useful structures to discover the periodic patterns efficiently. The suffix tree (Rasheed et al. 2011; Nishi et al. 2013) is built to find frequent repeat patterns. SMAC (Huang and Chang 2005) uses a vertical database which is more efficient than a horizontal database to store the time series data. Elfeky (Elfeky et al. 2005a) uses binary representation to transform the time series data into binary vectors which allows efficient pattern matching. More researchers mined periodic patterns on several different structures of time series data. Pujeri and Karthik (Pujeri and Karthik 2012) applied periodic pattern mining in multiple time series sequences and proposed an efficient and flexible constraint based periodicity mining technique.

The suffix tree based algorithm (Rasheed and Alhajj 2010, 2008; Xylogiannopoulos et al. 2012) can determine the pattern and its positions of occurrences in a time series. The collection of occurrence positions is called occurrence vector. Rasheed et al. (Rasheed et al. 2011) proposed a suffix-tree-based noise-resilient (STNR) algorithm to

mine periodic patterns and find symbol, sequence (partial periodic), and segment (full cycle) periodicity in time series data. They also proposed to mine periodic patterns with periodicity in subsection of time series rather than entire time series by specifying start and end position. STNR algorithm can avoid multiple types of noises like replacement, insertion, deletion, or any mixture of these types of noise. They also introduced the concept of time tolerance which lets the noise-resilient algorithm be more flexible. However, STNR cannot determine the occurrence vectors of inner patterns, i.e., the patterns with wildcard between two items and thus STNR is not able to mine such periodic patterns. Nishi et al. (Nishi et al. 2013) also observed this problem of STNR algorithm. They proposed an approach to define a maximum event skipping threshold to constrain the number of wildcards between any two items and follow Apriori-like level-by-level sequential pattern mining method to generate the complex inner pattern. However, using the maximum event skipping threshold, the algorithm cannot mine all possible periodic inner patterns. Furthermore, the Apriori-based level-by-level method may generate many candidates and joining patterns can be time-consuming for large periods.

As mentioned above, our proposed algorithm, MIPPS is designed to mine different kinds of periodic patterns simultaneously without the maximum event skipping threshold.

In recent years, some research works have developed methods for different variants of periodic pattern mining. Among these, some research have focused on periodic-frequent pattern mining (Tanbeer et al. 2009; Kiran and Kitsuregawa 2014; Kiran et al. 2015, 2017; Nofong and Wondoh 2019). The study in (Li et al. 2015) focuses on mining periodicity from incomplete observations. Another study (Yuan et al. 2017) presents a method to discover periodic mobility patterns. The study in (Chanda et al. 2017) presents a framework for mining weighted periodic patterns. Similarly, the studies (Chen et al. 2019; Yuan et al. 2019) discusses some other issues related to periodicity. Nevertheless, the problem definitions of these works are different from the problem definition discussed in this study.

3 Preliminaries

Definition 1 Time series A time series $T\{i_1, i_2, i_3, ..., i_n\}$ of length n is a set of n values containing an item i_j at every time point t_j. Each item can be viewed as an event and the time series can be viewed as an event sequence.

Definition 2 Periodic pattern Periodic pattern $P\{i_1, i_2, i_3, ..., i_x\}$ is an ordered list of items of length x that repeats itself in the time series.

A periodic pattern may contain a wildcard ($*$) in place of an item. A wildcard, $*$, represents any item or unimportant item at a particular time position.

Definition 3 Full periodic pattern A periodic pattern $P\{i_1, i_2, i_3, ..., i_x\}$ is a full periodic pattern if all items in P are selected from the possible set of items in the time series. That is to say, a full periodic pattern does not contain any wildcard, $*$.

Definition 4 Partial periodic pattern A periodic pattern $P\{i_1, i_2, i_3, ..., i_x\}$ is called a partial periodic pattern, $if\ \exists\ j, 1 \leq j \leq x, i_j = *$.

Partial periodic patterns contain at least one wildcard. We never have any pattern which starts with a wildcard because it is identical to other patterns with a shifted starting position in the time series (Yang et al. 2003).

Definition 5 Inner periodic pattern A partial periodic pattern $P\{i_1, i_2, i_3, ..., i_x\}$ is called an inner periodic pattern, if some wildcards occur between two items and the last item i_x is not a wildcard. E.g., $\{a, *, b\}$, $\{a, *, b, *, c\}$.

Definition 6 Tail periodic pattern A partial periodic pattern $P\{i_1, i_2, i_3, ..., i_x\}$ is called a tail periodic pattern, if the last item i_x is a wildcard. E.g., $\{a, b, *\}$, $\{a, *, c, *\}$.

Definition 7 The least-period-of-the-pattern, $|P|$ The least-period-of-the-pattern $|P|$ is the size of pattern P, i.e., number of items in the pattern P including wildcard. The least period of pattern is the minimum distance between two occurrences of the periodic pattern. Since, in this work, we care about only non-overlapping periodic patterns, the least period of periodic pattern is equal to the length of the pattern. For example, the least-period-of-the-pattern value for the patterns $\{a, b, c\}$, $\{a, *, c\}$ and $\{a, *, *\}$ is 3.

Definition 8 Perfect periodicity Given a time series $T\{i_1, i_2, i_3, ..., i_n\}$ and a pattern $P\{i_1, i_2, i_3, ..., i_x\}$, let $T'\{i'_1, i'_2, i'_3, ..., i'_m\}$ be a segment of time series T where $1 \leq m \leq n$, and $O\{t_1, t_2, t_3, ..., t_k\}$ be an ordered collection of starting time point of each occurrence of P in T'. P has perfect periodicity in T' if $\forall j, 1 \leq j \leq k-1, t_{j+1} - t_j = |P|$.

Definition 9 Imperfect periodicity Given a time series $T\{i_1, i_2, i_3, ..., i_n\}$ and a pattern $P\{i_1, i_2, i_3, ..., i_x\}$, let $T'\{i'_1, i'_2, i'_3, ..., i'_m\}$ be a segment of time series T where $1 \leq m \leq n$, and $O\{t_1, t_2, t_3, ..., t_k\}$ be an ordered collection of starting time point of each occurrence of P in T'. P has imperfect periodicity in T' if $\forall j, 1 \leq j \leq k-1, mod(t_{j+1} - t_j, |P|) = 0$ and $\exists j, 1 \leq j \leq k-1, t_{j+1} - t_j > |P|$.

In other words, in imperfect periodicity at least one of the expected occurrences is missing.

Definition 10 Asynchronous periodicity Given a time series $T\{i_1, i_2, i_3, ..., i_n\}$, a pattern $P\{i_1, i_2, i_3, ..., i_x\}$ and a gap β, let $T'\{i'_1, i'_2, i'_3, ..., i'_m\}$ be a segment of time series T where $1 \leq m \leq n$, and $O\{t_1, t_2, t_3, ..., t_k\}$ be an ordered collection of starting time point of each occurrence of P in T'. P has asynchronous periodicity in T' if $\forall j, 1 \leq j \leq k-1, |P| \leq t_{j+1} - t_j \leq |P| + \beta$.

For example, if a time series T is *ababcabdabab* and the gap is 1, there is an asynchronous periodic pattern "ab" with period 2 that occurs periodically at positions {1, 3, 6, 9, 11}. In this example, after position 3, next occurrence is at 6 which is different than the expected position 3+2=5 or 5+2=7 according to period=2 that makes this pattern asynchronous periodic pattern. Note that, asynchronous periodicity does not allow overlapping between the two successive neighbor patterns, which is different than the periodicity with time tolerance (Rasheed et al. 2011). To find all the occurrences of frequent periodic patterns with asynchronous periodicity, we do not use the gap parameter in our algorithm so that all possible gaps are examined. Nevertheless, this parameter can be specified by the user for our algorithm if needed.

Definition 11 Maximum support of perfect periodicity($|P|$) The Maximum support of perfect periodicity($|P|$) is the maximum number of all possible occurrences of the $|P|$-period pattern with perfect periodicity in time series and is denoted by $\lfloor \frac{the\ length\ of\ time\ series}{|P|} \rfloor$

Definition 12 Actual Support of the $|P|$-period periodic pattern The support of the $|P|$-period periodic pattern is the number of occurrences of $|P|$-period pattern with a specific periodicity in a time series.

Definition 13 Confidence ($|P|$) The confidence of the $|P|$-periodic pattern is denoted by $\frac{Actual\ support\ of\ the\ periodic\ pattern}{Max\_Support\ of\ perfect\ periodicity(|P|)}$

Confidence is a formula to weigh the quality of a periodic pattern in the time series. In general, given a user defined minimum confidence, when the confidence ($|P|$) of periodic pattern is larger than or equal to the minimum confidence, the periodic pattern is called a frequent periodic pattern.

The actual support of the periodic pattern may have different values on the different periodicities or different sizes of period. In order to determine whether the periodic pattern meets the minimum confidence threshold or not, the minimum confidence can be converted to the minimum support.

Definition 14 Minimum support ($|P|$) **and Minimum repeat times** α $Minimum\ support = \lfloor Max\_support\_of\_perfect\_periodicity(|P|) \times Minimum\_Confidence \rfloor$

We observed that the maximum support of perfect periodicity changes with the size of period. When the period is large, the maximum support of perfect periodicity($|P|$) is small. Hence, patterns appearing few times can also satisfy the minimum support ($|P|$). For example, if the length of time series is 100 and the minimum confidence is 0.2, when the period of periodic pattern is 20, a pattern appearing only one time satisfies the minimum confidence. In order to avoid this problem, we define a threshold named *minimum repeat times* α. Any frequent periodic pattern should satisfy both minimum support ($|P|$) and α.

4 Algorithm MIPPS

We propose a novel algorithm to mine full and partial (inner and tail) periodic patterns with different periodicities (perfect, imperfect, and asynchronous) simultaneously. In the first phase, we build a suffix tree without suffix link which we describe in Section 4.1. Then in the second phase, we need to find different kinds of patterns and their occurrence vectors containing the information of all the positions where the pattern appears in the time series. We propose an incremental propagation generator to get different kinds of patterns and their occurrence vectors. Finally, we mine periodic patterns by detecting periods on occurrence vectors and by detecting different periodicities simultaneously. The basic procedures of MIPPS algorithm are shown in Fig. 1. We describe the details of MIPPS in following subsections.

MIPPS algorithm
1. Build the suffix tree without suffix link
2. Start from the root and get the initial template pattern
3. Breadth-first search to traverse all accessible edges of branch
3.1. Get an item from the labels of current edge and produce patterns
- Prune and reduce redundant generated pattern on the edge list
- Store pattern on the current edges
- Store inner pattern and the longest full pattern on the candidate list
4. Detect periodic pattern on candidate list
4.1. Prune pattern while detecting periodic pattern on candidate list
4.2. Detect all kinds of periodic patterns on the pattern of candidate list
- Use optimization Strategies while detecting periodic pattern
4.3. Clear the candidate list
5. Repeat steps 3-4 until the template pattern is equal to the |max|-period pattern

Fig. 1 MIPPS algorithm

4.1 Suffix tree-based data structure

In this paper, we propose a novel suffix tree-based algorithm MIPPS. There are three kinds of nodes in the suffix tree, say root, internal and leaf. The root node is the root of suffix tree containing nothing. A suffix of a time series is represented as a path from the root to a leaf. Internal nodes are in the path from the root and leaves. Except the root node, every node has an edge above itself and each edge has a label representing items in a suffix. A suffix tree can effectively find a substring which can be regarded as a recurrent pattern in time series data. The occurrence positions of the recurrent pattern are numbered on the leaves below the internal nodes. The collection of occurrence positions is called an occurrence vector. The occurrence vectors of the internal nodes of upper levels contain the occurrence vectors of each internal node of the lower levels under its hierarchy. The downward closure property of occurrence vector in suffix tree is useful to devise mining strategy. For a substring example, Fig. 2 shows a suffix tree of the time series bababcbc$. The occurrence vector of "b" is {1, 3, 5, 7} and the occurrence vector of "bab" is {1, 3}. The size of the occurrence vector of "bab" is 2, representing the number of positions in the time series where the pattern occurs. If the size of the occurrence vector of the pattern is not less than the minimum support, the pattern is potential to be a frequent periodic pattern.

When we build a suffix tree, we cannot get the occurrence vector of the internal nodes directly. The previous work (Rasheed et al. 2011) uses the Ukkonen's algorithm (Ukkonen 1995) to build the suffix tree and traverses the tree using a bottom-up traversal to get the occurrence vector of the internal nodes. In this paper, we also use the Ukkonen's algorithms to build the suffix tree but do not use suffix link. The building time of a suffix tree with suffix link is linear, $O(m)$, where m is the length of time series. Without suffix link the building time is $O(2m)$ (Smyth and Smyth 2003). The building time without suffix link is higher, but we can get the occurrence vector of internal nodes directly without the additional bottom-up traversal of the suffix tree. In addition, the bottom-up traversal algorithm needs to sort occurrence vectors of the internal nodes, but our strategy does not have this requirement. Figure 3 shows an example of suffix tree with occurrence vectors stored in the nodes.

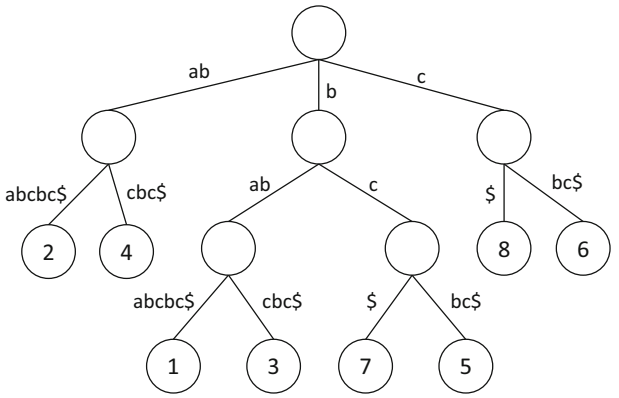
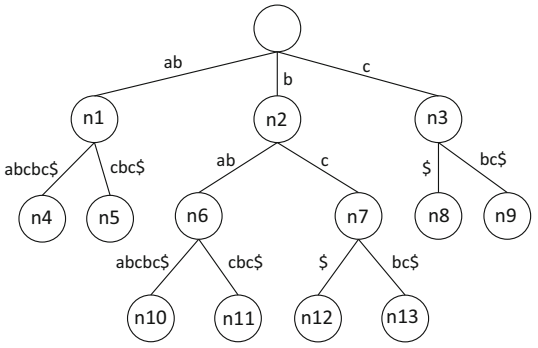


Fig. 2 The suffix tree of the String bababcbc$

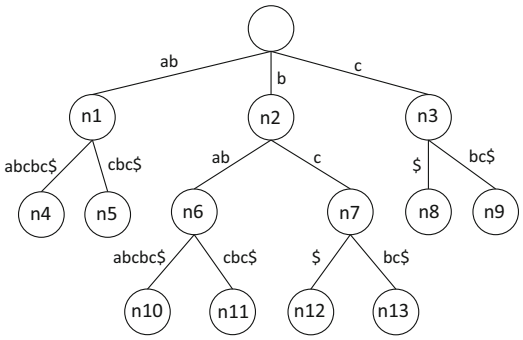


Node	Occ. Vector
n1	[2, 4]
n2	[1, 3, 5, 7]
n3	[6, 8]
n4	[2]
n5	[4]
n6	[1, 3]
n7	[5, 7]
n8	[8]
n9	[6]
n10	[1]
n11	[3]
n12	[7]
n13	[5]

Fig. 3 The suffix tree of the String bababcbc$ with occurrence vector in the nodes

The strategy is that while finding the path of a new suffix string, the reference of every internal node on the path is stored. If the end position of the path is reached, the occurrence vectors of internal nodes are updated. In MIPPS algorithm, this strategy is more suitable than using the bottom-up traversal of suffix tree. MIPPS is designed for a top-down traversal because there are some advantages and techniques to prune redundant generated patterns and periods for detecting periodic pattern. We discuss about these strategies in later sections.

However, for very large sequence data, storing the list of integers as occurrence vectors in the nodes is not efficient in terms of memory usage. Therefore, we devise a strategy to use boolean vectors in the nodes in place of integer vectors for occurrence information. The occurrence vector of the level-1 nodes (i.e., children nodes directly under root) contain the all the occurrence positions for the subtree. The occurrence vector of other nodes in subtree are subset of the occurrence vector of the level-1 node in the suffix tree. Therefore, we can store the occurrence vector of level-1 nodes in a separate list and we can store the index along with boolean vector in the nodes. Boolean vector contains either ‘1’ or ‘0’ boolean value for each number in the occurrence vector. Figure 4 shows an example of this strategy, where in a separate list



Index	Occ. Vector
0	[2, 4]
1	[1, 3, 5, 7]
2	[6, 8]

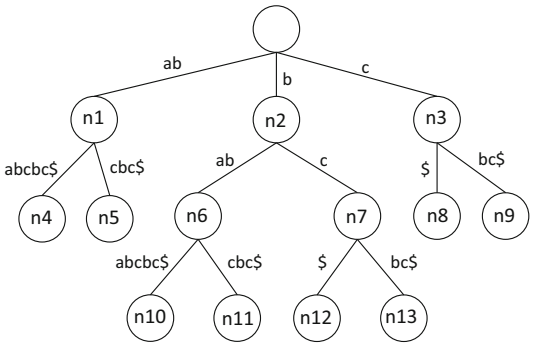
Node	Occ. Info.
n1	0, [1, 1]
n2	1, [1, 1, 1, 1]
n3	2, [1, 1]
n4	0, [1, 0]
n5	0, [0, 1]
n6	1, [1, 1, 0, 0]
n7	1, [0, 0, 1, 1]
n8	2, [0, 1]
n9	2, [1, 0]
n10	1, [1, 0, 0, 0]
n11	1, [0, 1, 0, 0]
n12	1, [0, 0, 0, 1]
n13	1, [0, 0, 1, 0]

Fig. 4 The suffix tree of the String bababcbc$ with occurrence vector using boolean vector in the nodes

we store the occurrence vector of the three level-1 nodes. To access the occurrence vector from the separate list, we store the index and boolean vector as occurrence information in the nodes.

Above mentioned strategy saves memory usage for suffix tree with occurrence vectors. However, for very large sequence data, many nodes in the suffix tree have to store too many leading and trailing zeros in the boolean vectors. To save more memory, we do not store leading and trailing zeros. Therefore, we store another integer in the occurrence information of the nodes which represent the position of first '1' in the boolean vector, i.e., index inside the occurrence vector of level-1 node. Figure 5 shows an example of this strategy. In the occurrence information of nodes, we have three fields; first is the integer representing the index of main list where actual occurrence vectors for level-1 nodes are stored, second is the integer representing the start position or start index in the occurrence vector represented by first field, third is the boolean vector without leading and trailing zeros. In addition, since boolean vector of level-1 nodes contains only 1s for all the positions in the occurrence vector, we store just a single boolean value '1' in the boolean vector to save memory. For level-1 nodes we can simply retrieve the complete occurrence vector from main list using the index value. This strategy is effective in terms of memory usage and it helps to retrieve the actual occurrence vector of the nodes efficiently.

A suffix tree has exactly m leaves numbered from 1 to m but a suffix string may be a prefix of other suffix strings which makes the suffix tree lose some numbered leaves. To avoid this situation, an end marker symbol (usually \$) is added at the end of string S. Furthermore, no two edges starting out of a node can have string-labels beginning with the same character, which ensures each sub-tree of the suffix tree is independent. Therefore, we can split the suffix tree from the first level and run MIPPS algorithm on each sub-tree. In Fig. 2, the suffix tree can be split into three sub trees that start from "ab", "b" and "c" respectively.



Index	Occ. Vector
0	[2, 4]
1	[1, 3, 5, 7]
2	[6, 8]

Node	Occ. Info.
n1	0, 0, [1]
n2	1, 0, [1]
n3	2, 0, [1]
n4	0, 0, [1]
n5	0, 1, [1]
n6	1, 0, [1, 1]
n7	1, 2, [1, 1]
n8	2, 1, [1]
n9	2, 0, [1]
n10	1, 0, [1]
n11	1, 1, [1]
n12	1, 3, [1]
n13	1, 2, [1]

Fig. 5 The suffix tree of the String bababcbc$ with occurrence vector using boolean vector without leading and trailing zeros in the nodes

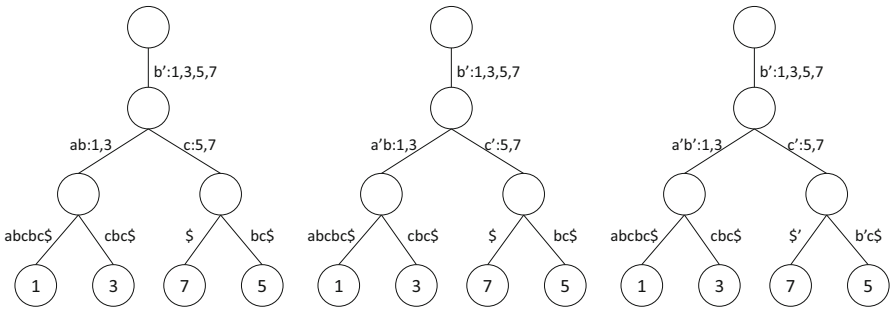
In order to mine full, inner, and tail periodic patterns, obtaining only the occurrence vector of full pattern is not enough. In the next subsection, we introduce an incremental propagation generator that can obtain the occurrence vectors of all these specified patterns.

4.2 Incremental propagation generator

According to the previous definitions, periodic patterns are classified into three types, say full, tail, and inner patterns. These patterns can be found by traversing the suffix tree built in the first phase. The path of a full pattern is unique in the suffix tree. Therefore, full patterns and their occurrence vectors can be found as substrings by following paths from the root to any item of the label (label may contain multiple items) on the edge. Inner pattern has wildcards that allow it to present at multiple paths in the suffix tree. If we want to obtain an occurrence vector of an inner pattern, we need to find all possible paths of the inner pattern. Finally, tail patterns can be extracted by finding a new period larger than the previous period of full and inner patterns. The following subsections explain the details of the incremental propagation generator.

4.2.1 Generate candidate patterns and their occurrence vectors

The initial template pattern is the label of the unique internal node at the first level of the sub tree. For example, in Fig. 2, there are three sub-trees and their initial templates are “ab”, “b” and “c” respectively. The first step of the incremental propagation generator produces the initial template pattern and it is also the first full pattern. Then, the



(a) first step of traversal **(b)** second step of traversal **(c)** third step of traversal

Fig. 6 The results of each traversal

incremental propagation generator extends the template pattern by appending an item or a wildcard (∗).

We use a breadth-first search to traverse all accessible edges below the internal node that was traversed previously. For each edge, current patterns are generated and stored in a list associated with the edge. The current pattern is generated from each template pattern by appending a wildcard (∗) or one item taken from the label on the edge. The item taken from the label on edge is marked as used. In Fig. 6, we use the apostrophe (') symbol to mark items as used. Each current pattern is a template pattern for extending the periodic pattern. For example, Fig. 6a is a sub-tree of Fig. 2 and the initial template pattern is b. Next, in Fig. 6b, current patterns of the left edge are produced, say ba and $b*$, and current patterns of the right edge are bc and $b*$. These patterns are also called candidate periodic patterns. For the known accessible edges, we only need to extend the template pattern of the edge which has an unmarked item in the label. If all items of the label on the edge have been marked as used, we traverse down to the next level and visit all new accessible edges. New accessible edges get the template patterns from their parent. Moreover, if the next item is $ on the label, we stop generating any pattern from this edge because $ is the end marker. For example, in Fig. 6b, item b of the left label $a'b$ has not been used. Therefore, we take item b from this label. On the other hand, the item of the right label c' is marked as used. Therefore, we go down to the next level and take one item from label bc\$ and \$. In Fig. 6c, the current patterns of the edge with label $a'b'$ are bab, $b * b$, $ba*$ and $b * *$. Current patterns of the edge with label $b'c$\$ are bcb, $b * b$, $bc*$ and $b * *$.

In the template patterns, one wildcard (∗) is added to generate tail patterns and one item from the label of the edge is added to generate full and inner patterns. The full pattern is found from the unique substring whose path starts from the root of the suffix tree. Therefore, the occurrence vector of the full pattern is the occurrence vector of the internal node below the edge. However, this is not applicable to the inner pattern because the inner pattern has wildcards which make it possible to exist on multiple paths in the suffix tree. In order to ensure that all paths containing the inner pattern are visited, the incremental propagation generator adopts breadth-first search which reaches all paths for the inner pattern. For example, in Fig. 6c, the occurrence vector of $b * b$ is obtained after traversing all current branches. The paths bab {1, 3} and bcb

{5} have a common inner pattern $b * b$ whose occurrence vector is {1, 3, 5} obtained from joining the occurrence vectors {1, 3} and {5}. Finally, the tail pattern is generated by appending one wildcard at the end of any pattern. Therefore, the occurrence vector of the tail pattern is equal to the occurrence vector of the template pattern from which the tail pattern is extended. For example, in the Fig. 6b, the occurrence vector of tail pattern $b*$ is equal to the occurrence vector of its parent b.

The incremental propagation generator continues to use breadth-first search to traverse all accessible edges and extends template patterns until the template pattern becomes the $|max|$-period pattern. $|max|$-period is the maximum possible period for patterns. It is restricted by the user defined threshold α which requires the pattern to appear the minimum repeat times (α) in a time series. Therefore, the maximum possible period is calculated by $\lfloor \frac{the\ length\ of\ time\ seris}{\alpha} \rfloor$.

4.2.2 Pruning strategies used while generating candidate patterns on the edge

(a) **Reduce generating full patterns:** If the size of the occurrence vector of the edge is smaller than the minimum repeat times(α), the corresponding periodic pattern cannot be frequent. Therefore, the candidate full pattern is not generated on this edge. In the suffix tree, there is only one path containing the specific full pattern. Due to the downward closure property, the size of the occurrence vector of extended patterns cannot be more than the size of the occurrence vector of the template pattern. Therefore, we do not need to generate all full patterns down below this pattern.

(b) **Prune useless candidate inner patterns:** Sometimes the candidate inner pattern does not have other paths to support its existence. The candidate inner pattern P is useless, if there exists a candidate inner pattern P' which has less number of wildcards than P and has the same occurrence vector as P. When two candidate inner patterns have the same occurrence vector on the same edge, the pattern with a larger number of wildcards is useless. Such useless candidate patterns are not extended further and are deleted. We remove such useless candidate inner patterns to avoid generating useless candidate periodic patterns and template patterns.

4.3 Mining different kinds of periodic patterns simultaneously

As already stated in the previous sections, we can get all possible patterns and their occurrence vectors. The next step is mining periodic patterns based on the occurrence vectors of patterns in the candidate list. This step is divided into two sub-procedures. The first sub-procedure detects all possible periods which may have periodic patterns. The other sub-procedure classifies the periodic patterns into periodicities in which they belong. MIPPS algorithm can do both sub-procedures simultaneously. Before mining periodic patterns on candidate patterns, we use some pruning strategies to prune candidate patterns. We explain these strategies in Section 4.3.1.

4.3.1 Pruning candidate patterns

Before mining periodic patterns of the three kinds of patterns and their occurrence vectors, we observe two properties about full and tail patterns that helps us to make strategies to detect periodic patterns only on inner patterns and the longest full pattern of the edge. We do not need to detect periodic patterns on tail and sub full patterns. In addition, we observe another property with the minimum repeat times(α) that can prune patterns while detecting periodic patterns.

Strategy 1 ***For full patterns, mine the periodic patterns from the occurrence vector of the longest full pattern of the edge.***

If an edge of the suffix tree contains multiple items, all the full patterns generated from the same edge get the same occurrence vector from the internal node below the edge. Therefore, the full patterns which end on the same edge but at different positions have the same occurrence vector. For example, in Fig. 2, full pattern ba and bab have the same occurrence vector $\{1, 3\}$. The substring which ends at the last position of the label on the edge is the longest substring at this edge, i.e., the longest full pattern on the edge. The generated candidate full patterns of the edge are sub-patterns of the longest full pattern, i.e., sub full patterns. Therefore, we do not mine periodic patterns on the generated candidate sub full patterns. We only need to mine periodic patterns on the longest full pattern of the edge. Furthermore, this strategy also avoids discovering some redundant periodic patterns. For example, the longest full pattern is abc and its sub full pattern is ab and they have the same occurrence vector $\{1, 4, 7\}$. When the period is three, the periodic pattern $ab*$ is a redundant periodic pattern because there is a better periodic pattern abc. In order to get the longest full pattern conveniently, our strategy is to get the longest full pattern and its occurrence vector when we examine a new edge. The longest full pattern is generated by appending the label of the edge to the longest full pattern of the parent edge. We ignore mining periodic patterns on sub full patterns without losing any interesting periodic sub full patterns. We explain more about this in Sect. 4.3.2.

Strategy 2 ***Discover periodic tail patterns by discovering longer periods of the full and inner patterns.***

The tail pattern is generated by adding one more wildcard at the end of any pattern. In other words, a wildcard added at the end of pattern can be regarded as an extra time point in a longer period. Therefore, when we detect a longer period than the least period of pattern, a wildcard can be automatically added on tail. For example, the pattern b has the least-period-of-pattern of the pattern b is 1. If we can find a period 2 for pattern b, the periodic tail pattern $b*$ is also discovered. We can discover the periodic tail patterns by detecting different periods on full and inner patterns. Nevertheless, we still generate the tail patterns and its occurrence vector on the edge to use it as the template pattern for further extension.

Strategy 3 ***The candidate patterns having the size of the occurrence vector less than the minimum repeat times*** (α) ***are pruned.***

The candidate patterns which have the size of the occurrence vectors less than the minimum repeat times(α) are deleted because such patterns and their extensions cannot become a periodic pattern satisfying α. However, we cannot delete the pattern which has the size of the occurrence vector less than the minimum support ($|P|$) because minimum support ($|P|$) changes with the size of the period. The current generated $|P|$-period pattern is infrequent according to the minimum support of the current period. However, this does not imply that the extended $|P+1|$-period pattern is also infrequent.

According to the properties mentioned above, we only mine periodic patterns on inner patterns and the longest full pattern of the edge whose size of the occurrence vector is more than the minimum repeat times (α).

4.3.2 Discovering all possible periods

The periodic patterns of different periods and different periodicities can be discovered from a candidate pattern. Periodic tail patterns can be discovered when the periodic full and inner patterns are detected with longer periods than the current pattern length. For both cases, we need to inspect all possible periods to find all possible periodic patterns.

The least-period-of-pattern $|P|$ is the least period that can include all items of the pattern. Therefore, we detect the periodic pattern starting from the least period. However, this property can apply on only inner patterns and cannot work on full patterns. In Strategy 1, we have discussed that to prune the redundant generated full pattern, only the longest full pattern needs to be mined for the periodic pattern. For this reason, the least period of the longest full pattern should start from the length of the longest full pattern of the parent edge plus one. This strategy can avoid losing periodic patterns that are sub patterns of the longest full pattern. For example in the Fig. 6b, to detect sub patterns from the longest full pattern bab, the period starts from 2, i.e. the length of the longest pattern of the parent edge b plus one. In this way, the periodic pattern $ba\{1, 3\}$ is found, which is a sub pattern of the longest full pattern bab.

In addition, the maximum period is restricted by α. Therefore, the maximum possible period is calculated by $\lfloor \frac{the\ length\ of\ time\ seris}{\alpha} \rfloor$. All possible periods of the pattern lies in the range starting from the least period to the maximum period.

4.3.3 Discovering periodicity connection

A periodic pattern is discovered by connecting more time points from the occurrence vector of the same pattern. Between any two successive neighbor time points, there exists a periodicity connection. In order to clearly recognize those periodicity connections, two values are used to analyze two successive neighbor time points. The first value is *Remainder*, which is the result after comparing the moduli of two successive time points by the same period, i.e., mod period. The second value is *Difference*, which

Table 2 The category of detected periodicity

Category	Remainder	Difference	Periodicity
1	Equal	Zero	Perfect
2	Equal	More than zero	Imperfect
3	Unequal	More than zero (less than or equal to gap if it is specified)	Asynchronous

is the difference of two successive time points minus the period. These two values can describe what kind of periodicity connection exists in the two successive time points. For example, perfect periodicity only appears when *Remainder* is equal and *Difference* is zero. The other results of asynchronous and imperfect periodicities are shown in Table 2.

Difference can also help us to know whether two successive neighbor patterns are overlapping or there is a gap between two successive neighbor patterns. Overlapping patterns share at least one common time point in the time series. Therefore, there is no periodicity connection. If *Difference* is less than zero, there are overlapping patterns. If *Difference* of two time points is larger than zero, there is a gap. If *Remainder* is equal and there is a gap, there is an imperfect periodicity. When *Remainder* is equal, the gap is synchronous, or in other words the gap is equal to $period * y$ where integer $y \geq 1$. Similarly, if *Remainder* is unequal and there is a gap, there is an asynchronous periodicity.

Now, we introduce how to use these periodicity connection categories to mine full, inner, and tail periodic patterns with perfect, imperfect, and asynchronous periodicity simultaneously. While examining the periodicity connection category between two successive time points of the occurrence vector, we may get different category of periodicity connection at different positions in the occurrence vector. When the *category* transforms from one to another, a periodic pattern with a previous periodicity *category* is discovered. Perfect periodicity may become imperfect periodicity and imperfect periodicity may change to asynchronous periodicity. Table 3 shows all possible cases for periodicity changes while examining the successive occurrence time points of a periodic pattern for periodicity. If the periodicity connection *category* does not transform to another *category*, the periodic pattern remains the same as the current periodicity. As mentioned above, the *category* will determine how many possible periodicities there are. If the *category* is perfect periodicity, there are possibly three kinds of periodic pattern which may exist, say perfect, imperfect and asynchronous. On the other hand, if the *category* is imperfect periodicity or asynchronous periodicity, there are only two possible kinds of periodic pattern based on periodicity because imperfect periodicity or asynchronous periodicity does not transform to perfect periodicity.

Given a period and the occurrence vector of a pattern, we detect the periodic pattern that starts from the first time point of the occurrence vector of the pattern and calculate *Remainder* and *Difference* to know the first periodicity connection *category* as in Table 2. This can help us to find the different periodic patterns when the *category* transforms to other periodicity connection *category*. We continue to calculate

Table 3 Periodicity transform table

Cases	Transformation of periodicity category
Case 1	Perfect → Imperfect (Category 1 → Category 2)
Case 2	Perfect → Asynchronous (Category 1 → Category 3)
Case 3	Imperfect → Asynchronous (Category 2 → Category 3)
Case 4	Asynchronous contains Imperfect (Category 3 contains Category 2)

periodicity connection *category* to mine periodic patterns until all time points of the occurrence vector are inspected. The start time point of different kinds of periodic patterns will be discovered simultaneously.

For example, given a minimum support 3, the pattern *ab* and its occurrence vector is {1, 3, 5, 9, 14}, we want to find periodic patterns of period 2. The *category* is 1 from time point 1 to 3. The *category* of the time point from 3 to 5 is the same as the current *category* that lets the periodicity of the periodic pattern continue. At the time point from 5 to 9, the *category* changes to 2. Therefore, the first periodic pattern is discovered at {1, 3, 5} and the repeat times is 3. It is a periodic pattern with perfect periodicity because the current *category* from time point 1 to 5 is 1, i.e., perfect periodicity. From time point 9, the current periodicity *category* is changed to 2, i.e., imperfect periodicity. Continuing from time point 9, we find that the *category* becomes 3 at time point 14, i.e., asynchronous periodicity. The second periodic pattern is discovered at {1, 3, 5, 9} and the repeat times is 4 with imperfect periodicity. Since the current periodicity *category* is changed to 3 at time point 14 and all time points are visited, the last periodic pattern is discovered with asynchronous periodicity at {1, 3, 5, 9, 14}.

To detect all possible periodic patterns, any time point of the occurrence vector is possibly a start time point of the periodic pattern. Therefore, we need to consider every time point of the occurrence vector as a start time point. We use the periodicity connection *category* to mine periodic patterns, which allows us to try the time point which is not connected by perfect periodicity simultaneously. There are four important strategies that can help us to select a start time point to mine periodic patterns based on periodicity.

Strategy 1 When we find a time position range which has a perfect periodicity, we skip to test with another start time point within this range because the perfect periodicity is the strongest connection for this range. Therefore, it includes the periodic patterns that start from other time points. From the previous example, the periodic patterns starting from time point 1 includes the periodic patterns that start from time point 3.

Strategy 2 If two successive neighbor time points are overlapping or there is a gap, the later time point is the next start time point used to detect periodic patterns. For example, pattern *aba* has occurrence vector {1, 3, 6, 9, 14, 17, 20}. For the period 3, the first start time point is 1 and the next start time point is 3 because of the overlapping patterns of time point 1 and time point 3. After time point 3, the next start time point is 14 because there is a gap 2 between time points 9 and 14. The overlapping does not allow the periodic pattern to include the later time point for the current periodicity. Therefore, to avoid losing any periodic pattern that starts from this time point, we need

to check for all possible periodic patterns. For example, pattern *bab* has occurrence vector {1, 3, 7, 10}. When the period is 3, time point 1 and 3 are overlapping. If the start time is 1, there is an imperfect periodicity at {1, 7, 10}. If the start time is 3, there is an asynchronous periodicity at {3, 7, 10}. Similarly, gap will generate different patterns starting at different time points. For example, pattern *ab* with the occurrence vector {1, 5, 7, 9} and the period is 2. When the start time point is 1, there is an imperfect periodicity at {1, 5, 7, 9}. Meanwhile, there is also a perfect periodicity at {5, 7, 9} that starts from time point 5.

Strategy 3 To avoid wasting time to find a periodic pattern that starts from a time point that cannot satisfy the minimum support, the difference between the size of the occurrence vector and the index of the start time point in occurrence vector should not be less than the minimum support, where index starts from 0. For example, the occurrence vector of *ab* is {2, 4, 8, 10, 12} and minimum support is 3. From the time point 10, the rest time points can be skipped since periodic patterns starting from those time points cannot satisfy the minimum support.

Strategy 4 We allow any gap in the imperfect periodicity or asynchronous periodicity. The imperfect periodicity in the periodic pattern that starts from the first time point will include other imperfect periodicity which starts from another time position for same period. The same scenario holds for asynchronous periodicity. Therefore, we only need to find the longest periodic patterns with imperfect periodicity and/or asynchronous periodicity that starts at the first time point of the occurrence vector.

4.4 Optimization strategies used while detecting periodic patterns

Strategy 1 Prune pattern while detecting periodic pattern:

If the size of the occurrence vector of the $|P|$-period pattern is less than the minimum support, we do not need to detect any periodic pattern of period P from this candidate pattern because none of them can be frequent.

Strategy 2 Prune periods while detecting periodic pattern:

Multiple periodic patterns with different periods can be discovered from a candidate pattern. Therefore, we should inspect all possible periods for the pattern. If the number of periods increase, the cost of time to mine periodic patterns increases too. Fortunately, MIPPS can minimize this issue. After examining the initial template pattern for all possible periods, the periods containing periodic patterns that satisfy minimum support are stored. The generated patterns after the initial template pattern only need to be examined for the periods which contain a frequent periodic pattern. Periods smaller than the least-period-of-the-pattern can be skipped. MIPPS works on each sub suffix tree. The occurrence vector of the initial template pattern contains all leaves of the sub suffix tree. Top down traversal ensures the initial template pattern generated from the suffix tree has downward closure property. The generated pattern after the initial template pattern cannot have a size of occurrence vector larger than the size of the occurrence vector of the initial template pattern. As mentioned above, the periods of frequent periodic patterns of later generated patterns are one of the periods which contain frequent periodic patterns from the initial template pattern. Moreover, the maximal period from those periods is the max period of the sub tree. If the template

pattern is extended to the $|max|$-period pattern, MIPPS stops, because we cannot have any frequent periodic pattern larger than the $|max|$-period pattern.

Strategy 3 Overlapping pruning theorem:

The size of the occurrence vector of a pattern is the number of time points in the occurrence vector. If the size of the occurrence vector of a pattern is not less than the minimum support, the pattern may be a frequent periodic pattern. However, the size of the occurrence vector does not provide information whether the two successive neighboring time points overlap or not. For example, the occurrence vector of pattern *aba* is {1, 4, 6, 10, 12} and the period is 3. The size of the occurrence vector of the pattern is 5. When the minimum support is 4, there are two possible start time points, 1 and 4, where the periodic pattern may satisfy the minimum support. However, there cannot be a frequent periodic pattern, because time points 4 and 10 overlap with 6 and 12 respectively, and only two time points can be chosen from these four time points. This kind of overlapping case allows us to stop detecting periodic pattern after time point 1. In order to detect the condition, we count the number of overlapping time points in the occurrence vector while detecting periods. There may be different overlaps according to different periods. Therefore, each detected period needs to be checked. When overlapping happens, we calculate the difference between the size of the occurrence vector and the index of the start time point in the occurrence vector minus the current count of overlapping. This calculated result allows to follow the following two corollaries.

- *Overlapping corollary 1* If the result is less than the minimum support, the later time points can be ignored to detect periodic pattern. This is because a periodic pattern that starts from the later time points cannot satisfy the minimum support.
- *Overlapping corollary 2* When the result is less than the minimum repeat times (α), the later periods can be ignored because patterns in those periods cannot satisfy the minimum repeat times.

The minimum support changes for different periods. We cannot ignore the later periods, when the result is less than the minimum support because there may be smaller minimum support when the period increases. However, the minimum repeat times (α) does not change. If the current period contains too many overlaps, the longer periods contain more overlaps. When the result is less than the minimum repeat times (α), we store the detected period as a *limited period*. Due to the downward closure property, the occurrence vector of the extended generated pattern is smaller than the occurrence vector of the template pattern. A *limited period* can restrict the extended generated pattern to be a $|limited\ period|$-period pattern. If the occurrence vector of the extended generated pattern is equal to the template pattern, overlaps still exist and due to the overlaps, the size of the occurrence vector cannot be more than the minimum repeat times (α). When the occurrence vector of the extended generated pattern does not have the overlap, the number of remaining occurrence time points still cannot be more than the minimum repeat times (α). As a result, if the period of the generated pattern is equal to the *limited period* of the template pattern, the generated pattern will not be checked for periodic patterns. The generated pattern inherits the *limited period* from the template pattern and it is useful to prune the later extended generated patterns too.

5 Experiments

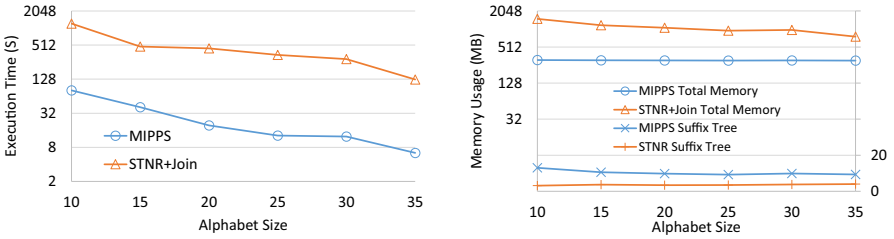
In this section, we compare the performance of MIPPS with another existing suffix tree based algorithm, STNR (Suffix-Tree-based Noise-Resilient algorithm), proposed in (Rasheed et al. 2011). We use synthetic data and real data in our experiments. We investigate the performance with different conditions such as the number of distinct items, the size of periods for periodic patterns and the different length of time series. Moreover, we also analyze memory usage for building suffix tree and discovering periodic patterns from suffix tree. We implemented the algorithms using Java programming language. The experiments are conducted on a computer with windows 10 64-bit operating system, an i7-3770 CPU of clock rate 3.40GHz, 32 GB of main memory, and OpenJDK 12. Execution time and memory usage are recorded using the Java API.

5.1 Synthetic data generation

For the purpose of performance evaluation, we generated synthetic time series data set consisting of $|N|$ distinct items, $|D|$ time points, $|K|$ patterns, $|L|$ size of period and $|G|$ max gap. In order to inspect the accuracy of the proposed algorithms, the synthetic data have three kinds of pattern (full, inner and tail pattern) and three kinds of periodicity (perfect, imperfect and asynchronous). A synthetic data set was generated as follows: First, we decide the period of a periodic pattern by normal distribution with the mean as $|L|$ and the standard deviation as 1. Then we use uniform distribution to generate one of three kinds of patterns. Then the above steps are repeated until $|K|$ periodic patterns are produced. Second, we decide the periodicity of the periodic pattern using uniform distribution for the three kinds of periodicities. The patterns of asynchronous periodicity are randomly inserted in the data. The interval between two successive neighbor patterns of perfect or imperfect periodic pattern have some limits. Therefore, these periodic patterns are inserted as a time segment. The interval of two successive neighbor patterns confirm the limit of periodicity in the time segment. The number of occurrences of the periodic pattern is calculated by $\lfloor \frac{|D|/|K|}{period\ of\ periodic\ pattern} \rfloor$. Finally, the minimum repeat times (α) for this synthetic dataset can be set as the lowest number of occurrences of periodic pattern. The minimum support for this synthetic dataset can be set as the value, $\frac{number\ of\ occurrence\ of\ periodic\ pattern}{(|D|/period\ of\ periodic\ pattern)}$.

5.2 Experiments on synthetic data

Before presenting the experimental results, we first describe how to implement STNR (Rasheed et al. 2011) to mine full, tail, and inner periodic patterns. Originally, STNR can find the full and tail periodic patterns. For the inner periodic pattern, STNR needs an additional technique to combine full and tail patterns with the same period and periodicity. The combining method is a depth first search which avoids generating too many redundant periodic patterns. The interval of two successive neighbors vary for different occurrences when the periodic pattern has imperfect or asynchronous periodicity which makes the occurrence vector of combined periodic inner pattern



(a) Execution Time **(b)** Memory Usage

Fig. 7 Execution time and memory usage on different number of distinct items

incorrect. Therefore, it needs to be checked again by scanning the database once. We call this implementation as 'STNR+Join' for reporting the experimental results and discussion.

Both MIPPS and STNR+Join algorithms find periodic patterns of all the possible periods according to minimum confidence and minimum repeat parameters. Period size parameter values specified in the experiments are only used for generating the synthetic datasets.

5.2.1 Varying number of distinct items

The first experiment investigates the performance on different number of distinct items, i.e., alphabet size. The parameters of synthetic data are set as time points = 10K, pattern = 5, size of period = 10 and max gap = 5.

The execution time and memory usage of this experiment are shown in Fig. 7a and b respectively. In Fig. 7b, the total memory usage values of both algorithms are shown on primary Y-axis, i.e., on the left side whereas memory usage of suffix tree of both algorithms are shown on secondary Y-axis, i.e., on the right side. The Y-axis of Fig. 7a and Primary Y-axis of Fig. 7b are in the log scale of base 2.

When the number of distinct items, i.e., alphabet size decreases in the dataset, the probability of repeat of items or patterns increases. Hence, the number of candidate patterns increase with the decrease of alphabet size. Therefore, execution time of both algorithms increases with the decrease of number of distinct items, i.e., alphabet size. The experimental results illustrates that MIPPS is efficient in comparison to STNR+Join in terms of execution time as well as memory usage. Since MIPPS stores occurrences with suffix tree, memory usage for suffix tree of STNR is less than that of MIPPS. However, In terms of total memory usage, MIPPS uses less memory than STNR+Join. MIPPS generates the promising candidates and finds full, tail, and inner patterns simultaneously while traversing the suffix tree. In addition, pruning strategies helps to prune the search space by pruning useless candidates and periods. However, STNR+Join can only find full and tail periodic patterns while traversing the suffix tree. In order to mine inner periodic patterns, STNR+Join depends on an additional technique to join periodic patterns of the same period and periodicity which is time consuming. Due to these reasons, MIPPS is efficient against STNR+Join in terms of execution time.

The memory usage of a suffix tree based algorithm can be divided into building suffix tree, storing all occurrence time points and storing candidates. Our building strategy does not need the suffix link. Therefore, MIPPS conserves a little memory. In order to get the occurrence vector of the edge conveniently and make MIPPS efficient, the occurrence vector is stored on the nodes for the edges above it, which makes MIPPS use more memory than STNR. For the candidates, MIPPS generates patterns by extending the last generated template patterns. Therefore, MIPPS only needs to store the last generated template patterns and the current generated candidates in each iteration. In order to mine full, tail, and inner periodic patterns, STNR+Join depends on an additional technique to join periodic patterns of the same period and periodicity. Therefore, STNR+Join needs to store all the generated candidates for periodic inner patterns along with the mined full and tail periodic patterns. Due to these reasons, STNR+Join requires more memory than MIPPS in terms of total memory usage.

5.2.2 Varying periods

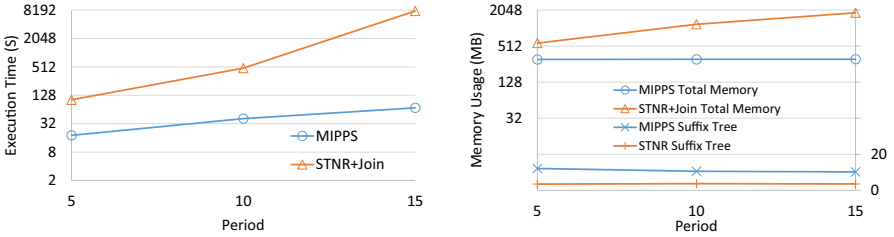
The second experiment is executed to show the performance on various period sizes. The parameters of the synthetic data are set as distinct items = 15, time points = 10K, pattern = 5, and max gap is the period size divided by 2. The synthetic datasets are generated using the period sizes 5, 10 and 15.

The execution time and memory usage on synthetic datasets generated using various period sizes are shown in Fig. 8a and b respectively. In Fig. 8b, the total memory usage values of both algorithms are shown on primary Y-axis, i.e., on the left side, whereas memory usage of suffix tree of both algorithms are shown on secondary Y-axis, i.e., on the right side. The Y-axis of Fig. 8a and Primary Y-axis of Fig. 8b are in the log scale of base 2.

The complexity of inner patterns usually increases when the size of period is large. MIPPS can handle complex patterns and still have better performance than STNR+Join. In STNR+Join, a large period of the pattern can include more wildcards and causes the combining method to spend more time for checking the combination of inner patterns. However, MIPPS prunes the useless candidates and periods while traversing the suffix tree for finding full, tail and inner periodic patterns. Therefore, MIPPS achieve better performance in comparison to STNR+Join in terms of execution time. In addition, MIPPS uses less memory than STNR+Join and performs better in terms of total memory usage as well. The memory usage of STNR+Join is significantly more when the period is large. The reason behind this is that small periodic patterns have very less inner patterns and the memory usage of candidates is low. When the period is larger, STNR+Join uses more memory since it generates many candidates for complex inner patterns and stores these candidates along with all the found full and tail periodic patterns.

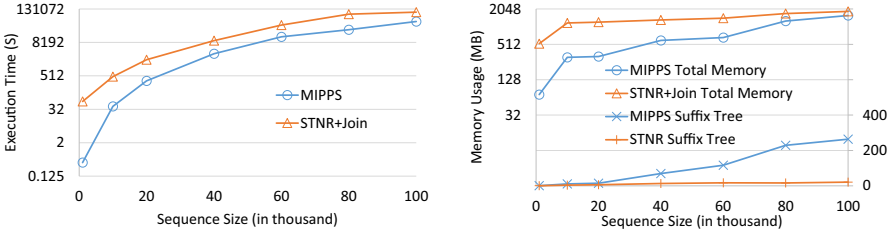
5.2.3 Different size of datasets

The third experiment is executed to illustrate the performance on different lengths of time series. The parameters of the synthetic data are set as distinct items = 15, period



(a) Execution Time **(b)** Memory Usage

Fig. 8 Execution time and memory usage on different period size



(a) Execution Time **(b)** Memory Usage

Fig. 9 Execution Time and Memory Usage on Different Length of Time Series

= 10, and pattern = 5. The datasets used for this experiments are of sequence sizes 1, 10, 20, 40, 60, 80, and 100K.

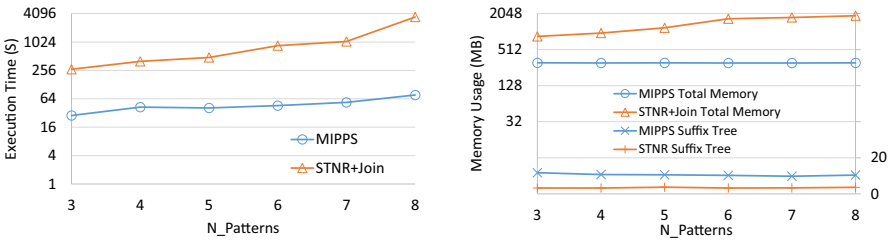
The execution time and memory usage of this experiments are shown in Fig. 9a and b respectively. In Fig. 9b, the total memory usage values of both algorithms are shown on primary Y-axis, i.e., on the left side, whereas memory usage of suffix tree of both algorithms are shown on secondary Y-axis, i.e., on the right side. The Y-axis of Figs. 9a and Primary Y-axis of Fig. 9b are in the log scale of base 2.

The execution time and total memory uses of both algorithm increases with the increase of sequence size. Nevertheless, MIPPS always has good performance in comparison to STNR+Join in terms of execution time. In addition, MIPPS uses less memory than STNR+Join in terms of total memory usage. When the length of time series is very large, the size of the occurrence vectors increases. Therefore, suffix tree of MIPPS uses more memory in comparison to STNR.

5.2.4 Different no. of patterns

The fourth experiment is executed to illustrate the performance on the datasets generated using varying number of patterns. The parameters of the synthetic data are set as distinct items = 15, period = 10, and sequence size 10K.

The execution time and memory usage of this experiments are shown in Fig. 10a and b respectively. In Fig. 10b, the total memory usage values of both algorithms are shown on primary Y-axis, i.e., on the left side, whereas memory usage of suffix tree of both algorithms are shown on secondary Y-axis, i.e., on the right side. The Y-axis of Fig. 10a and Primary Y-axis of Fig. 10b are in the log scale of base 2.



(a) Execution Time **(b)** Memory Usage

Fig. 10 Execution Time and Memory Usage on Different No. of Patterns

Table 4 Description of real biological data

ID (Name)	Sequence length
A2ASS6 (TITIN_MOUSE)	35,213
Q8WZ42 (TITIN_HUMAN)	34,350
P17437 (Skin secretory protein xP2)	439
P09593 (S-antigen protein)	375

The execution time of both algorithms increases with the increase of number of patterns used for generating dataset. However, the execution time of MIPPS increases slightly. When there are more frequent periodic patterns in dataset, the number of candidate inner patterns increases too. Thus, STNR+Join requires more time during join operation. Therefore, we can see the difference between MIPPS and STNR+Join in execution time when there is increase in number of patterns. In addition, MIPPS requires less memory in comparison to STNR+Join in terms of total memory usage.

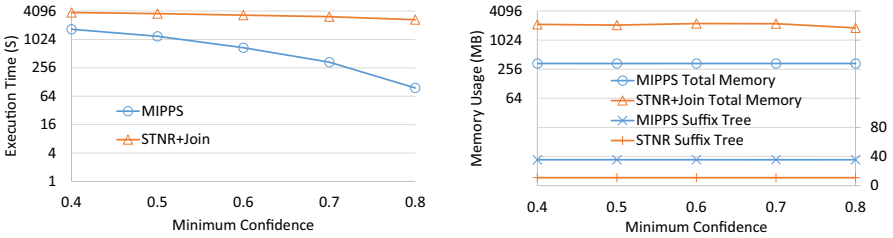
5.3 Experiments on real data

To compare the performance evaluation on real data, we perform experiments on the two real biological sequence data A2ASS6 and Q8WZ42, which are downloaded from UniProt website (https://www.uniprot.org/uniprot/). The description of these dataset are shown in Table 4.

5.3.1 Effect of varying minimum confidence

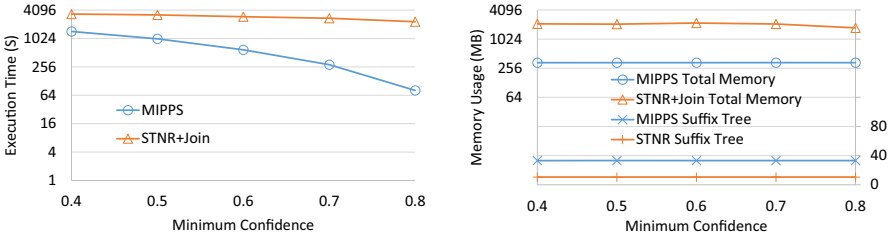
This experiment is executed to demonstrate the performance of both algorithms on real biological sequence dataset *A2ASS6* and *Q8WZ42* for various minimum confidence values. The minimum repeat value is set at 400 for this experiment.

The experimental results of this experiment on dataset *A2ASS6* and *Q8WZ42* are shown in Figs. 11 and 12 respectively. In Figs. 11b and 12b, the total memory usage values of both algorithms are shown on primary Y-axis, i.e., on the left side, whereas memory usage of suffix tree of both algorithms are shown on secondary Y-axis, i.e., on the right side. The Y-axis of Figs. 11a and 12a are in the log scale of base 2. Similarly, Primary Y-axis of Figs. 11b and 12b are in the log scale of base 2.



(a) Execution Time **(b)** Memory Usage

Fig. 11 Effect of varying minimum confidence on A2ASS6 dataset



(a) Execution Time **(b)** Memory Usage

Fig. 12 Effect of varying minimum confidence on Q8WZ42 dataset

The experimental results shown in Figs. 11 and 12 illustrate that MIPPS is efficient in comparison to STNR+Join in terms of both execution time and total memory usage. Because MIPPS prunes the candidates and periods at each level while traversing the suffix tree, MIPPS can prune more when minimum confidence increases. However, STNR+Join needs post-processing, i.e., join operation.

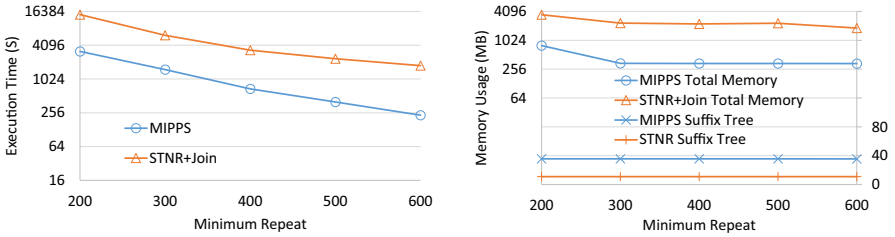
5.3.2 Effect of varying minimum repeat

This experiment is executed to demonstrate the performance of both algorithms on real biological sequence dataset *A2ASS6* and *Q8WZ42* for various minimum repeat values. Minimum confidence for this experiment is set at 0.6.

The experimental results of this experiment on dataset *A2ASS6* and *Q8WZ42* are shown in Figs. 13 and 14 respectively. In Figs. 13b and 14b, the total memory usage values of both algorithms are shown on primary Y-axis, i.e., on the left side, whereas memory usage of suffix tree of both algorithms are shown on secondary Y-axis, i.e., on the right side. The Y-axis of Figs. 13a and 14a are in the log scale of base 2. Similarly, Primary Y-axis of Figs. 13b and 14b are in the log scale of base 2.

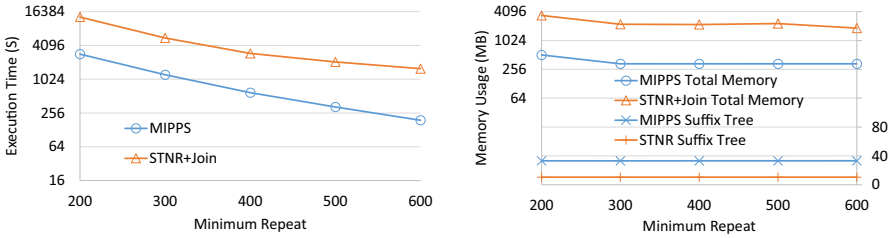
The experimental results shown in Figs. 11 and 12 illustrate that with decrease of minimum repeat execution time of both algorithms increase. However, MIPPS significantly outperforms STNR+Join in terms of both execution time and total memory usage.

The experiments on both synthetic datasets and real datasets illustrate the superiority of MIPPS over STNR+Join in terms of execution time as well as total memory usage.



(a) Execution Time **(b)** Memory Usage

Fig. 13 Effect of varying minimum confidence on A2ASS6 dataset



(a) Execution Time **(b)** Memory Usage

Fig. 14 Effect of varying minimum repeat on Q8WZ42 dataset

5.4 Use case on real biological data

We apply MIPPS on biological data. This is a practical application of periodic pattern mining to find recurrent patterns in DNA or protein sequences. DNA sequences are constructed using four bases represented by letters A, T, C, and G. Protein sequences are based on 20 amino acids. These sequences have their own specific properties. For example, periodic patterns are only found in a subsection of the sequence and do not span the entire sequence length. In addition, pattern occurrence may be shifted from the expected position. The biological data are from the PROSITE database of the ExPASy Molecular Biology Server (http://www.expasy.org/). According to Katti (Katti et al. 2000), the protein sequence *P17437* (Skin secretory protein xP2) has a known periodic pattern $\{APAPA**E**\}$ whose number of repeats is 25. When MIPPS is applied, the periodic pattern $\{APAPA**E**\}$ and the repeat number of 25 is found, which confirms that MIPPS can be applied towards real life data. In addition, MIPPS also finds that the $\{APAPA**E**\}$ pattern has 19 repeats with perfect periodicity starting at position 118. Moreover, other interesting periodic patterns with more than 25 repeats are found. The periodic pattern $\{APA***AP\}$ has 30 repeats and the periodic pattern $\{EG*AP*PA\}$ has 28 repeats. SMCA (Huang and Chang 2005) confirms the periodic pattern $\{APAPA**E**\}$ and further reports finding an interesting longer pattern $\{APAPAEGEAP\}$ with 11 repeats. MIPPS finds this pattern $\{APAPAEGEAP\}$ with more repeats, say 17 repeats. Some of the results of this experiment are shown in Table 5.

Table 5 Experimental results on biological data - P17437

Period	Pattern	Repeats	Occurrence positions
10	$APAPA**E**$	25	[44, 62, 80, 90, 100, 118, 128, 138, 148, 158, 168, 178, 188, 198, 208, 218, 228, 238, 248, 258, 268, 278, 288, 298, 336]
10	$APAPA**E**$	19 (Perfect)	Starting at position 118
8	$APA***AP$	30	[30, 38, 48, 56, 74, 84, 94, 104, 112, 122, 132, 142, 150, 160, 170, 180, 190, 200, 212, 222, 232, 242, 252, 262, 272, 282, 290, 300, 310, 328]
8	$APAEG**P$	29	[30, 38, 48, 56, 66, 84, 94, 104, 112, 122, 132, 142, 150, 170, 180, 190, 200, 212, 222, 232, 242, 252, 262, 272, 282, 290, 300, 310, 328]
8	$APAEG*AP$	28	[30, 38, 48, 56, 84, 94, 104, 112, 122, 132, 142, 150, 170, 180, 190, 200, 212, 222, 232, 242, 252, 262, 272, 282, 290, 300, 310, 328]
8	$EG*AP*PA$	28	[33, 41, 51, 59, 87, 97, 107, 115, 125, 135, 145, 153, 173, 183, 193, 203, 215, 225, 235, 245, 255, 265, 275, 285, 293, 303, 313, 331]
10	$APAPAEGEAP$	17	[46, 120, 130, 140, 168, 178, 188, 198, 210, 220, 230, 240, 250, 260, 270, 280, 298]

Table 6 Experimental results on biological data - P09593

Period	Pattern	Repeats	Occurrence positions
11	$KGTGGPGS***$	21	[113, 124, 135, 146, 157, 168, 179, 190, 201, 212, 223, 234, 245, 256, 267, 278, 289, 300, 316, 331, 346]
11	$KGTGGPGSE**$	20	[113, 124, 135, 146, 157, 168, 179, 190, 201, 212, 223, 234, 245, 256, 267, 278, 289, 300, 316, 331]
11	$E*PKGTG**G*$	21	[110, 121, 132, 143, 154, 165, 176, 187, 198, 209, 220, 231, 242, 253, 264, 275, 286, 297, 308, 328, 343]
11	$E*PKGTGGPGS$	20	[110, 121, 132, 143, 154, 165, 176, 187, 198, 209, 220, 231, 242, 253, 264, 275, 286, 297, 328, 343]
11	$EGPKGTGGPGS$	19	[110, 121, 132, 143, 154, 165, 176, 187, 198, 209, 220, 231, 242, 253, 264, 275, 297, 328, 343]
11	$PKGTGGPGS**$	21	[112, 123, 134, 145, 156, 167, 178, 189, 200, 211, 222, 233, 244, 255, 266, 277, 288, 299, 315, 330, 345]
11	$PKGTGGPGSE*$	20	[112, 123, 134, 145, 156, 167, 178, 189, 200, 211, 222, 233, 244, 255, 266, 277, 288, 299, 315, 330]
11	$GPKGTGGPGS*$	20	[111, 122, 133, 144, 155, 166, 177, 188, 199, 210, 221, 232, 243, 254, 265, 276, 298, 314, 329, 344]
11	$GPKGTGGPGSE$	19	[111, 122, 133, 144, 155, 166, 177, 188, 199, 210, 221, 232, 243, 254, 265, 276, 298, 314, 329]
11	$GGPGSE*PKGT$	19 (Perfect)	Starting at position 105
11	$GPGSE*PKGTG$	19 (Perfect)	Starting at position 106
11	$GGPGSEGPKGT$	18	[105, 116, 127, 138, 149, 160, 171, 182, 193, 204, 215, 226, 237, 248, 259, 270, 292, 303]
11	$GPGSEGPKGTG$	18	[106, 117, 128, 139, 150, 161, 172, 183, 194, 205, 216, 227, 238, 249, 260, 271, 293, 304]
11	$GGPGSEGPKGT$	16 (Perfect)	Starting at position 105
11	$GPGSEGPKGTG$	16 (Perfect)	Starting at position 106
11	$PGSEGPKGTGG$	16 (Perfect)	Starting at position 107
11	$GSEGPKGTGGP$	16 (Perfect)	Starting at position 108
11	$SEGPKGTGGPG$	16 (Perfect)	Starting at position 109
11	$EGPKGTGGPGS$	16 (Perfect)	Starting at position 110
11	$GPKGTGGPGSE$	16 (Perfect)	Starting at position 111

We also run MIPPS on the protein sequence *P09593* (S-antigen protein). (Katti et al. 2000) reported that $\{GGPGSEGPKGT\}$ pattern with period 11 has 19 repeats. As described in (Rasheed et al. 2011), a particular repeating pattern is only different by one amino acid at position 6 which has S instead of G. Similar to (Rasheed et al. 2011), MIPPS also finds this pattern with 18 repeats. (Rasheed et al. 2011) also reported that they found an interesting pattern $\{GPGSEGPKGTG\}$, which is a shifted version of the $\{GGPGSEGPKGT\}$ pattern with 18 repeats. MIPPS also finds this shifted version of the pattern. In addition, MIPPS also finds another interesting pattern $\{GPKGTGGPGSE\}$ with 19 repeats starting at position 111. The *P09593* protein sequence has the pattern $\{GGPGSEGPKGT\}$ with perfect periodicity and 16 repeats starting at position 105. MIPPS finds this as well as another 6 of its shifted versions with same periodicity and repeats, i.e., perfect periodicity and 16 repeats starting from the 106 to 111 positions. In addition, MIPPS finds some interesting periodic patterns with more than 19 repeats. Some of the results of period 11 from this experiment are shown in Table 6.

According to the above results, MIPPS not only finds patterns reported by previous literature, but also discovers some interesting novel periodic patterns in the real life protein sequences.

6 Conclusions

In this study, we proposed a suffix-tree based algorithm, Mining dIfferent kinds of Periodic Patterns Simultaneously (MIPPS), to find full, inner, and tail periodic patterns with perfect, imperfect and asynchronous periodicities simultaneously. Using top down traversal and the proposed incremental propagation generator, MIPPS can generate occurrence vectors of all the full, inner, and tail periodic patterns simultaneously. MIPPS use three properties that can prune redundant generated patterns to avoid producing useless patterns in the suffix tree. Then, MIPPS use a single method for periodicity detection, that can detect periodic patterns with perfect, imperfect, and asynchronous periodicities simultaneously. In addition, MIPPS uses some optimization strategies to make the periodicity detection more efficient. The results of several experiments show that MIPPS has good performance on the different conditions of synthetic data. Moreover, the experimental results on biological data illustrates that MIPPS algorithm can be applied to real life data and find more interesting novel patterns. There may be scalability issue for very very long dataset that have too many periods and periodicities. In this case, parallel and distributed algorithms may be suitable. We would like to work on these area in our future work.

References

Ahdesmäki M, Lähdesmäki H, Pearson R, Huttunen H, Yli-Harja O (2005) Robust detection of periodic time series measured from biological systems. BMC Bioinform 6(1):1–18. https://doi.org/10.1186/1471-2105-6-117

Berberidis C, Aref WG, Atallah M, Vlahavas I, Elmagarmid AK (2002) Multiple and partial periodicity mining in time series databases. In: Proceedings of the 15th European Conference on Artificial Intelligence, ECAI'2002, Lyon, France, July 2002, pp 370–374

Burkom HS, Murphy SP, Shmueli G (2007) Automated time series forecasting for biosurveillance. Stat Med 26(22):4202–4218. https://doi.org/10.1002/sim.2835

Cao H, Cheung DW, Mamoulis N (2004) Discovering partial periodic patterns in discrete data sequences. In: Pacific-Asia conference on knowledge discovery and data mining, pp 653–658, https://doi.org/10.1007/978-3-540-24775-3_77

Chanda AK, Ahmed CF, Samiullah M, Leung CK (2017) A new framework for mining weighted periodic patterns in time series databases. Expert Syst Appl 79:207–224. https://doi.org/10.1016/j.eswa.2017.02.028

Chen J, Li K, Rong H, Bilal K, Li K, Yu PS (2019) A periodicity-based parallel time series prediction algorithm in cloud computing environments. Inform Sci 496:506–537. https://doi.org/10.1016/j.ins.2018.06.045

Elfeky MG, Aref WG, Elmagarmid AK (2005a) Periodicity detection in time series databases. IEEE Trans Knowl Data Eng 17(7):875–887. https://doi.org/10.1109/TKDE.2005.114

Elfeky MG, Aref WG, Elmagarmid AK (2005b) Warp: time warping for periodicity detection. In: Fifth IEEE International Conference on Data Mining (ICDM'05), pp 8 pp.–, https://doi.org/10.1109/ICDM.2005.152

Esling P, Agon C (2012) Time-series data mining. ACM Comput Surv 45(1):1–34. https://doi.org/10.1145/2379776.2379788

Fu TC (2011) A review on time series data mining. Eng Appl Artificial Intell 24(1):164–181. https://doi.org/10.1016/j.engappai.2010.09.007

Glynn EF, Chen J, Mushegian AR (2005) Detecting periodic patterns in unevenly spaced gene expression time series using lomb-scargle periodograms. Bioinformatics 22(3):310–316. https://doi.org/10.1093/bioinformatics/bti789

Han J, Dong G, Yin Y (1999) Efficient mining of partial periodic patterns in time series database. In: Proceedings 15th International Conference on Data Engineering (Cat. No.99CB36337), pp 106–115, https://doi.org/10.1109/ICDE.1999.754913

He Z, Wang XS, Lee BS, Ling ACH (2008) Mining partial periodic correlations in time series. Knowl Inform Syst 15(1):31–54. https://doi.org/10.1007/s10115-006-0051-5

Huang KY, Chang CH (2005) Smca: a general model for mining asynchronous periodic patterns in temporal databases. IEEE Trans Knowl Data Eng 17(6):774–785. https://doi.org/10.1109/TKDE.2005.98

Katti MV, Sami-Subbu R, Ranjekar PK, Gupta VS (2000) Amino acid repeat patterns in protein sequences: their diversity and structural-functional implications. Protein Sci 9(6):1203–1209. https://doi.org/10.1110/ps.9.6.1203

Kiran RU, Kitsuregawa M (2014) Novel techniques to reduce search space in periodic-frequent pattern mining. In: International Conference on Database Systems for Advanced Applications, Springer International Publishing, Cham, pp 377–391, https://doi.org/10.1007/978-3-319-05813-9_25

Kiran RU, Shang H, Toyoda M, Kitsuregawa M (2015) Discovering recurring patterns in time series. In: Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23–27, 2015, pp 97–108. https://doi.org/10.5441/002/edbt.2015.10

Kiran RU, Shang H, Toyoda M, Kitsuregawa M (2017) Discovering partial periodic itemsets in temporal databases. In: Proceedings of the 29th International Conference on Scientific and Statistical Database Management, Association for Computing Machinery, New York, NY, USA, SSDBM '17, pp 1–6, doi:https://doi.org/10.1145/3085504.3085535

Li Z, Wang J, Han J (2015) Eperiodicity: mining event periodicity from incomplete observations. IEEE Trans Knowl Data Eng 27(5):1219–1232. https://doi.org/10.1109/TKDE.2014.2365801

Lin J, Keogh E, Wei L, Lonardi S (2007) Experiencing sax: a novel symbolic representation of time series. Data Min Knowl Discovery 15(2):107–144. https://doi.org/10.1007/s10618-007-0064-z

Lin T-H, Kaminski N, Bar-Joseph Z (2008) Alignment and classification of time series gene expression in clinical studies. Bioinformatics 24(13):147–155. https://doi.org/10.1093/bioinformatics/btn152

Nishi MA, Ahmed CF, Samiullah M, Jeong BS (2013) Effective periodic pattern mining in time series databases. Expert Syst Appl 40(8):3015–3027. https://doi.org/10.1016/j.eswa.2012.12.017

Nofong VM, Wondoh J (2019) Towards fast and memory efficient discovery of periodic frequent patterns. J Inform Telecommun 3(4):480–493. https://doi.org/10.1080/24751839.2019.1634868

Ouyang R, Ren L, Cheng W, Zhou C (2010) Similarity search and pattern discovery in hydrological time series data mining. Hydrol Process 24(9):1198–1210. https://doi.org/10.1002/hyp.7583
Ozden B, Ramaswamy S, Silberschatz A (1998) Cyclic association rules. In: Proceedings 14th International Conference on Data Engineering, pp 412–421, doi:10.1109/ICDE.1998.655804
Pierson E, Althoff T, Leskovec J (2018) Modeling individual cyclic variation in human behavior. In: Proceedings of the 2018 World Wide Web Conference, International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, WWW '18, pp 107–116, doi:10.1145/3178876.3186052
Pujeri RV, Karthik GM (2012) Constraint based periodicity mining in time series databases. Int J Comput Netw Inform Secur 4(10):37–46. https://doi.org/10.5815/ijcnis.2012.10.04
Rasheed F, Alhajj R (2008) Using suffix trees for periodicity detection in time series databases. In: 2008 4th International IEEE Conference Intelligent Systems, pp 11–8–11–13, doi:10.1109/IS.2008.4670501
Rasheed F, Alhajj R (2010) Stnr: a suffix tree based noise resilient algorithm for periodicity detection in time series databases. Appl Intell 32(3):267–278. https://doi.org/10.1007/s10489-008-0144-9
Rasheed F, Alshalalfa M, Alhajj R (2011) Efficient periodicity mining in time series databases using suffix trees. IEEE Trans Knowl Data Eng 23(1):79–94. https://doi.org/10.1109/TKDE.2010.76
Sheng C, Hsu W, Lee ML (2006) Mining dense periodic patterns in time series data. In: 22nd International Conference on Data Engineering (ICDE'06), pp 115–115, doi:10.1109/ICDE.2006.97
Sirisha G, Shashi M, Raju GP (2014) Periodic pattern mining–algorithms and applications. Global J Comp Sci Technol 13(13-C). https://computerresearch.org/index.php/computer/article/view/268
Smyth B, Smyth W (2003) Computing patterns in strings. Pearson Education, London
Song H, Li G (2008) Tourism demand modelling and forecasting-a review of recent research. Tour Manag 29(2):203–220. https://doi.org/10.1016/j.tourman.2007.07.016
Tanbeer SK, Ahmed CF, Jeong BS, Lee YK (2009) Discovering periodic-frequent patterns in transactional databases. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer Berlin Heidelberg, pp 242–253, doi:10.1007/978-3-642-01307-2_24
Ukkonen E (1995) On-line construction of suffix trees. Algorithmica 14(3):249–260. https://doi.org/10.1007/BF01206331
Weigend AS, Gershenfeld NA (1994) Time series prediction: forecasting the future and understanding the past. Addison-Wesley. https://dblp.org/rec/books/aw/WeigendG94.bib
Xylogiannopoulos KF, Karampelas P, Alhajj R (2012) Periodicity data mining in time series using suffix arrays. In: 2012 6th IEEE International Conference Intelligent Systems, pp 172–181, doi:10.1109/IS.2012.6335132
Yang J, Wang W, Yu PS (2003) Mining asynchronous periodic patterns in time series data. IEEE Trans Knowl Data Eng 15(3):613–628. https://doi.org/10.1109/TKDE.2003.1198394
Yuan H, Qian Y, Bai M (2019) Efficient mining of event periodicity in data series. In: International Conference on Database Systems for Advanced Applications, Springer, pp 124–139, doi:10.1007/978-3-030-18576-3_8
Yuan Q, Zhang W, Zhang C, Geng X, Cong G, Han J (2017) Pred: Periodic region detection for mobility modeling of social media users. In: Proceedings of the Tenth ACM International Conference on Web Search and Data Mining, Association for Computing Machinery, New York, NY, USA, WSDM '17, pp 263–272, doi:10.1145/3018661.3018680
Zhong S, Khoshgoftaar TM, Seliya N (2007) Clustering-based network intrusion detection. Int J Reliab Qual Safe Eng 14(02):169–187. https://doi.org/10.1142/S0218539307002568