This article appeared in a journal published by Elsevier. The attached copy is furnished to the author for internal non-commercial research and education use, including for instruction at the authors institution and sharing with colleagues.

Other uses, including reproduction and distribution, or selling or licensing copies, or posting to personal, institutional or third party websites are prohibited.

In most cases authors are permitted to post their version of the article (e.g. in Word or Tex form) to their personal website or institutional repository. Authors requiring further information regarding Elsevier’s archiving and manuscript policies are encouraged to visit:

http://www.elsevier.com/copyright
Effective periodic pattern mining in time series databases

Manziba Akanda Nishia, Chowdhury Farhan Ahmeda, Md. Samiullaha, Byeong-Soo Jeongb,a

a Department of Computer Science and Engineering, University of Dhaka, Bangladesh
b Department of Computer Engineering, Kyung Hee University, South Korea

ABSTRACT

The goal of analyzing a time series database is to find whether and how frequent a periodic pattern is repeated within the series. Periodic pattern mining is the problem that regards temporal regularity. However, most of the existing algorithms have a major limitation in mining interesting patterns of users interest, that is, they can mine patterns of specific length with all the events sequentially one after another in exact positions within this pattern. Though there are certain scenarios where a pattern can be flexible, that is, it may be interesting and can be mined by neglecting any number of unimportant events in between important events with variable length of the pattern. Moreover, existing algorithms can detect only specific type of periodicity in various time series databases and require the interaction from user to determine periodicity. In this paper, we have proposed an algorithm for the periodic pattern mining in time series databases which does not rely on the user for the period value or period type of the pattern and can detect all types of periodic patterns at the same time, indeed these flexibilities are missing in existing algorithms. The proposed algorithm facilitates the user to generate different kinds of patterns by skipping intermediate events in a time series database and find out the periodicity of the patterns within the database. It is an improvement over the generating pattern using suffix tree, because suffix tree based algorithms have weakness in this particular area of pattern generation. Comparing with the existing algorithms, the proposed algorithm improves generating different kinds of interesting patterns and detects whether the generated pattern is periodic or not. We have tested the performance of our algorithm on both synthetic and real life data from different domains and found a large number of interesting event sequences which were missing in existing algorithms and the proposed algorithm was efficient enough in generating and detecting periodicity of flexible patterns on both types of data.

1. Introduction

Data mining refers to extracting or mining knowledge from large amounts of data. However, Pattern mining is one of the most important areas in data mining that includes frequent pattern (Agrawal & Srikant, 1994; Ahmed, Tanbeer, Jeong, & Choi, 2012; Ahmed, Tanbeer, Jeong, Lee, & Choi, 2012), sequential pattern (Agrawal & Srikant, 1995; Pei et al., 2004; Srikant & Agrawal, 1996; Zaki, 2001), inter transaction pattern (Lu, Han, & Feng, 1998; Tung, Lu, Han, & Feng, 1999), periodic patterns (Tanbeer, Ahmed, Jeong, & Lee, 2009) and episode mining (Mannila, Toivonen, & Inkeri Verkamo, 1997, 1995). The periodic pattern mining is performed on a time series database, which is a collection of data values gathered generally at uniform interval of time to reflect certain behavior of an entity.

Periodicity detection is a process for finding temporal regularities within the time series database (Rasheed, Alshalalfa, & Alhajj, 2007, 2011). Periodicity detection or periodic pattern mining has a number of applications, such as prediction, forecasting, detection of unusual activities, etc. The research work is related to the periodic pattern mining in time series database. It is an interesting data mining problem to search for the periodic pattern in time series database. In the existing algorithms, for the periodic pattern mining within time series databases, few user inputs must be entered, that is either user has to specify the period value for which the periodicity has to detect or the types of the periodic patterns have to be specified. That means, the user has to provide input to the algorithm specifying whether the algorithm will detect symbol, segment or full cycle periodic pattern.

The approaches used in the existing periodic pattern mining algorithms, have several limitations. For instance, suffix tree based algorithm (Rasheed, Al-Shalalfa, & Alhajj, 2011) has a limitation, that is, if we use the suffix tree to generate patterns and detect periodicity, we will fail to generate some flexible and interesting patterns which the proposed algorithm tried to overcome.
Moreover, using suffix tree, it is not possible to skip a particular character in a generated pattern where the pattern is a combination of two characters and each of the character is the representation of each of the independent events in a time series database. To get a clear idea consider the scenario where a time series database is represented as $T = \{abcd\}$. Using a suffix tree, we can only generate the eight types of patterns “abcdabcd”, “bcabcd”, “cdabcd”, “dabcd”, “abad”, “bed”, “ed”, and “d”. Suppose, user is interested to generate a pattern by skipping any intermediate character which stands for an unimportant event. Therefore, if the user wants that $a$, $b$ and $d$ will be in the first, second and fourth position respectively. Moreover, the user wants the third positioned character to be represented as don’t care event. By considering these issues, the pattern will be $(ab\bar{d})$. Throughout the paper, $\bar{a}$ represents don’t care event unless stated otherwise. User wants to skip this event and wants to consider the later events in the same generated pattern.

However, using a suffix tree, we cannot generate patterns like $(ab\bar{d})$ due to its inability to skip any intermediate event in a generated pattern. Hence, it is impossible to generate the pattern using a suffix tree which is the combination of the important and unimportant events from user’s point of view. As a consequence, from $T = \{abcd\}$, the proposed algorithm will result that $(ab\bar{d})$ occurs in the position $[0,4]$ in the form of $(abcd)$ and $(\bar{a}bc)$ respectively. In most of the existing algorithms, users have to enter the period value which is not an efficient process. Accordingly, user may miss any period value which is important. As well as the existing algorithms have limitations in detecting the type of periodicity they detect. Some can detect only symbol periodicity, and some detect only sequence or partial periodicity, while others detect only full cycle or segment periodicity.

As a motivating real life example, consider a company’s working hour database for employees, where the total working hour is divided into four slots, two hour in each slot per day. The company employees need to log-in at arrival and log-out at their leaving time. The log-in and log-out hour is stored in the database in order to calculate working hours of employees. An analogous database is shown in Table 1, which can be considered as a time series database with periodicity value $p = 4$. As a consequence, if the system administrator is asked by a manager of the company to calculate the total working hour of any employee. Then the computation can be done by searching the time series database by a pattern similar to $(a(x)^p)x$ where $a$ represents log-in time of an employee and $x$ stands for log-out time. Moreover, $\bar{x}$ represents the don’t care time instances within the working hour for that day, where the intermediate time instances are duty slots of that day and as the working hour can be varied, then $0 < Y \leq 2$. Therefore, actual mining patterns can be of the form $(ax)$, $(\bar{a}x)$ and $(a\bar{x}x)$. By searching such patterns through the whole database, we can perform the computation. This type of search can also be applied in predicting the behavior of that specific employee. Searching this type of time series database for such periodic pattern is not possible with existing algorithms.

These facts and the limitations of existing algorithms motivated us in developing a method of mining such interesting periodic patterns from time series database. To the best of our knowledge, this is a new approach for periodic pattern mining. The contributions of our work can be summarized as follows:

- Introducing a new method of periodicity mining in time series databases to generate patterns by adding extra flexibilities for the user to facilitate the discovery for those patterns which are generated by skipping intermediate events.
- The major drawback of all the existing algorithms about the rigidity of the patterns has been eliminated.
- The formation of few innovative measurements and criteria so that the overall intricacy of generation of the concluding significant patterns can be diminished.
- Discovery of all the three types of periodicity in one run, that is, the symbol, sequence and segment periodicity in more flexible and proficient way.
- Better user interaction is provided where users can provide period value and number of intermediate events that should be ignored in constructing flexible patterns.

2. Background study and related works

Mining periodic patterns in temporal databases is an important data mining problem in many applications (Rasheed et al., 2011). Periodicity detection is used in predicting future events or trends in time series databases; it is a crucial step towards better and more effective decision making (Rasheed & Alhajj, 2010). Research direction has focused either on devising general techniques for discovering potential periods (Elfeky, Aref, & Elmagarmid, 2004; Han, Dong, & Yin, 1999; Ma & Hellerstein, 2001; Toshniwal & Joshi, 2005) or devising special techniques for specific periodicity mining problems (Kolpakov & Kucherov, 1999; Rasheed & Alhajj, 2010). Both approaches require multiple phases over the time series database in order to output the periodic patterns. Previous works (Aref, Elfeky, & Elmagarmid, 2004; Han, Dong, & Yin, 1999; Ma & Hellerstein, 2001; Toshniwal & Joshi, 2005) devised methods to discover potential periods from the entire time series database.

Periodicity detection is useful to predict the behavior and the future trends of the time series database (Weigend & Gershenfeld, 1994). Such as, periodic pattern mining is proved to be useful in predicting the stock price movement, computer network fault analysis and detection of security breach, earth-quake prediction, and gene expression analysis (Glynn, Chen, & Mushegian, 2006). However, full cyclic pattern was first studied in Ozden, Ramsamy, and Silberschatz (1998). The input data to Ozden et al. (1998) is a set of transactions, each of which consists of a set of items. Periodicity mining allows an energy company to analyze power consumption patterns and predict periods of high and low usage so that proper planning may take place.

2.1. Preliminaries

This section introduces few terminologies required to get a clear perception on time series databases and to define algorithms for mining periodic patterns.
Definition 1 (Time series database). A time series database is a set of observations taken at specified times, usually at ‘equal intervals’. Mathematically a time series database is defined by a set of values \( Y_1, Y_2, \ldots, Y_n \) of a variable \( Y \) at times \( t_1, t_2, \ldots, t_n \). Thus, the relation among the variable values and time values can be defined as \( Y = F(t) \).

Definition 2 (Periodic pattern). An ordered list of events repeats itself in the event sequence is termed as periodic pattern.

As an example, in the event sequence “bbaa abaa acab abdd”, the pattern “ab” is a periodic pattern where the period value \( p = 4 \) and starting position = 4.

Definition 3 (Periodic pattern mining). Periodic pattern mining refers to the mining of patterns which are periodic within the time series database. In other words, it is a process of finding or extracting knowledge whether a given series, or a pattern within the series, is repeating itself at regular intervals or not. Periodicity detection is a process for finding temporal regularities within the time series database.

In general, three types of periodic patterns can be detected in a time series database:

- symbol periodicity
- sequence periodicity or partial periodic patterns
- segment or full-cycle periodicity

Definition 4 (Symbol, partial and full cycle periodicity). A time series database is said to have symbol periodicity if at least one symbol is repeated periodically. Similarly, a pattern consisting of more than one symbol may be periodic in a time series database, and this leads to partial periodic patterns. Finally, if the whole time series database can mostly be represented as a repetition of a pattern or segment, then this type of periodicity is called segment or full-cycle periodicity.

As an example, consider a time series database \( T = \{ abd \, acb \, aba \, ab \} \), where symbol/event ‘a’ is periodic with periodicity \( p = 3 \), starting at position zero (\( \text{stPos} = 0 \)). As well as, in a time series database \( T = \{ bbaa \, abda \, abda \} \), the sequence \( ab \) is periodic with periodicity \( p = 4 \), starting at position 4 (\( \text{stPos} = 4 \)); and the partial periodic pattern \( ab = \) exists in \( T \), where ‘\( \_ \)’ denotes any symbol representing don’t care event.

Definition 5 (Occurrence vector). Occurrence vector is a list of the index positions at which any substring exists in the original string. Therefore, the occurrence vector of a pattern within a string can also be defined as the list of all the index positions of the pattern where this sequence or pattern appears within the string.

As an illustration, consider the string \( T = \{ abc \, abcd \} \) is representing a time series database. Then the positions of each of the characters in the string is \( \{ a = 0, b = 1, c = 2, d = 3 \} \). So the occurrence vector for the pattern “ab” is \( \{0,4\} \). As the index positions of the substring “ab” is 0 and 4 within the string T.

Definition 6 (Confidence). The confidence of a periodic pattern \( X \), occurring in time series \( T \), is the ratio of its actual periodicity to its expected perfect periodicity.

Formally, the confidence of pattern \( X \) with periodicity \( p \) starting at position \( \text{stPos} \) is defined as:

\[
\text{conf}(p, \text{stPos}, X) = \frac{\text{Actual_Periodicity}(p, \text{stPos}, X)}{\text{Perfect_Periodicity}(p, \text{stPos}, X)}
\]

where

\[
\text{Perfect_Periodicity}(p, \text{stPos}, X) = \frac{|T| - \text{stPos} + 1}{p}
\]

Actual_Periodicity \((p, \text{stPos}, X)\) is computed by counting (starting at \( \text{stPos} \) and repeatedly jumping by \( p \) positions) the number of occurrences of \( X \) in \( T \).

2.2. Existing algorithms

Recently, there are few algorithms (e.g., Elfeky et al., 2005a; Elfeky, Aref, & Elmagarmid, 2005b; Indyk, Koudas, & Muthukrishnan, 2000) which look for all possible periods by considering the range \( 2 \leq p \leq n/2 \) where \( n \) is the number of transactions and \( p \) is the period value of the time series database. In Indyk et al. (2000), Indyk et al. have addressed this problem under the name periodic trends and have developed an \( O(n \log n) \) time algorithm (Elfeky et al., 2005b). One of the earliest best known works in this category has been developed by Elfeky et al. (2005b). They proposed two separate algorithms to detect symbol and segment periodicity in time series database. Their first algorithm (CONV) is based on the convolution technique with reported complexity of \( O(n \log n) \). Although their algorithm works well with data sets having perfect periodicity, it fails to perform well when the time series database contains insertion and deletion noise. Realizing the need to work in the presence of noise, Elfeky et al. later presented an \( O(n^2) \) algorithm (WARP) (Elfeky et al., 2005a), which performs well in the presence of insertion and deletion noise.

WARP uses the time warping technique to accommodate insertion or deletion noise in the data. Besides having \( O(n^2) \) complexity, however, WARP can only detect segment periodicity; it cannot find symbol or sequence periodicity. Also, both CONV and WARP can detect periodicity which last till the very end of the time series database, i.e., they cannot detect patterns which are periodic only in a subsection of the time series database. Sheng, Hsu, and Lee (2006) developed an algorithm which is based on Han et al.’s (1999) ParPer algorithm to detect periodic patterns in a section of the time series database; their algorithm utilizes optimization steps to find dense periodic areas in the time series database. However, their algorithm, being based on ParPer, requires the user to provide the expected period value. ParPer runs in linear \( O(n) \) time for a given period value, which is very difficult to provide. However, its complexity would increase to \( O(n^2) \) time if it is to be augmented to look for all possible periods. Also, ParPer can only detect partial periodic patterns; i.e., it cannot detect symbol and sequence periodicity.

2.2.1. Efficient periodicity mining in time series databases using suffix trees

The most recent algorithm for the periodic pattern mining in time series database is periodic pattern mining using suffix tree (Rasheed et al., 2011). The algorithm involves two phases. In the first phase, it builds a tree, which is known as suffix tree for the time series database and in the second phase, it uses the suffix tree to calculate the periodicity of various patterns in the time series database (Rasheed et al., 2011). The suffix tree for the string “ababababab” is shown in Fig. 1. Here the starting position of each of the patterns are shown in corresponding node.

- **First phase: suffix-tree-based representation**

  Given a time series database that is encoded as a string “ababababab”, where $\$ denotes end marker for the string; it is a unique symbol that does not appear anywhere in the string. Fig. 1 shows a suffix tree for the string. The path from the root to any leaf represents a suffix for the string. Since a string of length \( n \) can have exactly \( n \) suffixes, the suffix tree for a string also contains exactly \( n \) leaves. Each edge is labeled by the string that
it represents. Each leaf node holds a number that represents the starting position of the suffix when traversing from the root to that leaf. Each intermediate node holds a number which is the length of the substring, formed during the traversal from the root to that intermediate node. Each intermediate edge reads a string (from the root to that edge), which is repeated at least twice in the original string. These intermediate edges form the basis of the suffix tree based algorithm (Rasheed et al., 2011), presented in the next section.

- **Second phase: periodicity detection using suffix-tree**
  Once the tree as in Fig. 1, is constructed, the algorithm traverses the tree in bottom-up order to construct so called occurrence vector for each edge connecting an internal node to its parent. It starts with nodes having only leaf nodes as children; each such node passes the values of its children (leaf nodes) to the edge connecting it to its parent node. The values are used by the latter edges to create its occurrence vector (denoted occur_vec in the algorithm).

  The occurrence vector of edge $e$ contains index positions at which the substring from the root to edge $e$ exist in the original string. Later it considers each node $v$ having a mixture of leaf and non leaf nodes as children. The occurrence vector of the edge (s) connecting $v$ to its non leaf child node (s) and the value (s) coming from its leaf child node (s). Finally, until it reaches all direct children of the root, it recursively considers each node $u$ having only non leaf children.

  The occurrence vector of the edge connecting $u$ to its parent node is constructed by combining the occurrence vector (s) of the edge (s) connecting $v$ to its non leaf child node (s) and the value (s) coming from its leaf child node (s). Applying this bottom-up traversal process on the suffix tree shown in Fig. 1, will produce the occurrence vectors reported in Fig. 2. The tree traversal process is implemented using the non recursive explicit stack-based algorithm presented in Al-Rawi, Lansari, and Bouslama (2003), which prevents the program from throwing the stack-overflow-exception (Rasheed et al., 2011).

In Fig. 3, we have provided the pseudo-code of the existing, suffix tree based Periodicity Detection Algorithm (Rasheed et al., 2011).

### 3. The proposed approach

As we have discussed in Section 1, the proposed algorithm can detect periodicity of some exclusive and interesting patterns, those are missing in existing algorithms. It uses the similar data structure which is used in the sequential pattern mining algorithm (Zaki, 2001), but the application domain in this case is time series database. To facilitate such flexibilities we have defined several important terminologies. We have also extensively analyzed various facts and proposed an approach to perform such exclusive periodicity detection task. In this section, we have also shown how to apply discretization technique and then how to mine the string found by applying discretization technique on time series database. Finally, we have analyzed our proposed algorithm.

#### 3.1. Problem definition

We have defined several vital terminologies in this section to effectively define our proposed algorithm along with the actual problem formulation of our proposed work, that is, **Effective Periodic Pattern Mining**.

**Definition 7** (Maximum event skipping threshold, $\theta$). The maximum event skipping threshold value is the number up to which the user can skip the intermediate events between any two specific/interesting events. User can generate patterns containing $n < \theta$ number of don’t care events in between two important events, where $n$ is the number of the skipping events or don’t care events.

As an example, if maximum event skipping threshold value, $\theta = 3$, then user can generate patterns containing only 1, 2 or 3 number of don’t care events in between any two important events. Hence, $\{a=a+x\}$ is a valid pattern where $\{a=a+x\}$ is invalid. Because in between [a] and [x] user skips 4 events.

**Definition 8** (Pattern length, $L$). The length of a pattern is the number of events in that particular pattern.
As an example, the length of the pattern \([a\ldots x]\) and \([abc\ldots dx]\) is 5.

**Definition 9** (Difference vector). Difference vector of a pattern, \([p_0,p_1,\ldots,p_{n-1}]\) with sub-pattern \(p_1=[p_0,p_1,\ldots,p_{n-2}]\) and \(p_{n-1}=[p_{n-1}]\) and occurrence vector \([\text{Occ}_1,\text{Occ}_2,\ldots,\text{Occ}_{n-1}]\), is a list of values those represent the pairwise difference between the occurrence position of sub-patterns \(p_1\) and \(p_{n-1}\), which can be found from the occurrence vectors of the pattern \([p_0,p_1,\ldots,p_{n-1}]\).

As an example, in column-2 of Table 2, for a string \([ac\ldots cd\ldots dx]\), found from the Table 1 by applying discretization technique, we have three kinds of differences between the occurrence of \([a]\) and \([c]\). The first difference is 1. The position of the pattern \([ac]\) is \([0,1,2,4,5]\) and \([ac\ldots dx]\) is \([0,4,13,14]\). Hence, the difference between 0 and 1, 4 and 5, 13 and 14 is 1 as well as difference between 0 and 2 is 2. So the pattern is \([ac]\) and occurrence vector of \([ac]\) is \([0,4,13]\).

It implies that, we can check that \([ac]\) occurs in at position 0, 4 and 13 within the pattern \([ac\ldots dx]\). The difference between the occurrence position of \([a]\) and \([c]\) is 1, implicates that, in the string the position of \([c]\) is always 1 position later than \([a]\).

**Definition 10** (Effective periodic pattern mining). Given a sequence with \(n\) number of events, \(ES=(e_1,e_2,\ldots,e_n)\) of a time series database and a user specified maximum event skipping threshold, \(\theta\). We have to generate all the possible sub-sequences of events, \(SS=(e_i,e_{i+1},\ldots,e_{i+m})\subseteq ES\), where \(i \leq n\) with maximum \(\theta\) number of don’t care intermediate events within \(SS\).

In other words, we have to generate all possible event sub-sequences from the given event sequence and there could be don’t care events within the sub-sequences at any position and any number \(\leq \theta\) times.

### 3.2. Our proposed algorithm

The proposed algorithm discretizes the time series database as its initialization step, so that each of the independent event or transaction can be represented by a unique symbol. Next section defines the “Discretization technique” elaborately. The algorithm scans the entire database, discretizes it and converts into an event sequence or string of events. Then the generated string is searched for each of the individual event and its corresponding occurrence vector.

**3.2.1. Applying discretization technique**

The discretization can be thought as a mapping among the range of values of an entity and an ASCII character which represents a specific event and can be defined as a function of \(v(f)\).

### Table 2

<table>
<thead>
<tr>
<th>Patterns of length 1</th>
<th>Patterns of length 2</th>
<th>Patterns of Length 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a)</td>
<td>(b)</td>
<td>(c)</td>
</tr>
<tr>
<td>Occ_Vec</td>
<td>Occ_Vec</td>
<td>Occ_Vec</td>
</tr>
<tr>
<td>EID (a)</td>
<td>EID (b)</td>
<td>EID (c)</td>
</tr>
<tr>
<td>0</td>
<td>12</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>11</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>15</td>
</tr>
<tr>
<td>(ac)</td>
<td>(x)</td>
<td>(ac)</td>
</tr>
<tr>
<td>EID (a)</td>
<td>EID (c)</td>
<td>EID (x)</td>
</tr>
<tr>
<td>1</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>14</td>
<td>15</td>
<td>14</td>
</tr>
</tbody>
</table>
length two patterns, the proposed algorithm joins all pairs of single items if their event identifiers follow a sequential ordering. That is, the first item in the pair must occur as an event before its second item. Similarly, we can grow the length of item sets from length 2 to length 3. So, Event Id (X₁) < Event Id (X₂) < Event Id (X₃) < Event Id (X₄) will be maintained in generating occurrence vector of the pattern with length 4 if the joining condition is strictly followed. Here Xᵢ represents jth unique symbol in pattern length i. The joining condition can be expressed as follows.

\[ \text{EventId}(X_{i-1}) < \text{EventId}(X_i). \]  

(1)

Therefore, the occurrence vector in each pass will be

\[ [\text{EventId}(X_1), \text{EventId}(X_2), \text{EventId}(X_3), \ldots, \text{EventId}(X_n)] \]

As well as, the difference vector will be

\[ [\text{EventId}(X_2) - \text{EventId}(X_1), \ldots, \text{EventId}(X_n) - \text{EventId}(X_{n-1})] \]

It means, to find the event identifier of the length two patterns, we join event identifier of all pairs of single event. Then, the event identifier for each of the event in the pair is also joined only if the event identifier of the first event in pair is lower than the event identifier of the second event. That means, the proposed algorithm generates a pattern of length i in its ith pass and in generating length i patterns, length i – 1 patterns are joined with the condition in Eq. (1). Therefore, in pass 1, the algorithm generates length 1 patterns, in pass 2 it generates length 2 patterns. The proposed algorithm executes until the user defined number of pass is done.

For illustration consider Table 2, where the event identifier of \( \{a\} \) is [0,4,8,13] and \( \{c\} \) is [1,2,5,14]. Therefore, after joining we can find [0,1], [0,2], [0,5], [0,14], [1,1], [4,2], [4,5], [4,14], [8,1], [8,2], [8,5], [8,14], [13,1], [13,2], [13,5] and [13,14] for the pattern \( \{ac\} \). We will not pick [4,1], [4,2], [8,1], [8,2], [8,5], [13,1], [13,2] and [13,5] due to the former value of the pair is greater than the later. Here event_id of \( \{a\} \) should be less than event_id of \( \{c\} \). Now, we also have to eliminate those pairs having different elementary values greater than the period value. It complies that, [0,5], [0,14], [4,14] and [8,14] should also be removed from the event identifiers list. Hence, the accepted pairs are [0,1], [0,2], [4,5] and [13,14]. In a similar manner, after joining event identifiers of “c” and “x” we get the event identifier list for the pattern \( \{cx\} \). Accordingly, from these generated two length patterns, we can now mine the different types of patterns with don’t care events. Again, as shown in Table 2, we have joined the event identifiers of “ac” and “cx” to find the list of event identifiers for \( \{acx\} \), that is, [0,1,3], [0,2,3], [4,5,6] and [13,14,15]. In this process, we continue to generate the larger length patterns of length 4, 5, 6 and so on.

Our proposed algorithm joins two patterns of length i – 1 and forms new patterns of length i. This process begins with patterns of length 1 and continues till no more patterns can be generated or user provided requirement is met. In each phase, after any pattern is generated, the occurrence vector for that pattern needs to be computed. From the definition of occurrence vector, we have to search for the occurring positions of that pattern within main sequence to calculate its occurrence vector. This process requires extensive search of the patterns within the main event sequence which is time consuming. Following Lemma 1, resolves the problem and defines a way of calculating occurrence vectors for length i from already calculated length i – 1 or smaller patterns, which decreases the number of database scans.

**Lemma 1.** Let \( p = \{e_1; e_2; \ldots; e_n\} \in P \) is a mined pattern with the maximum event skipping threshold, \( \theta \) and the set of Occurrence Vectors \( = \{v_1; v_2; \ldots; v_n\} \) where \( \text{Occurrence Vector}_{v_i} = v_i \ (1 \leq i \leq n) \), then the Occurrence Vector \( p = v_i \), that is, the occurrence vector of the first event \( e_1 \), where \( P \) is a set of patterns.

**Proof.** As in Definition 5, the starting position/index for any sub-pattern within a pattern referred to as Occurrence Vector. Suppose, a sequential pattern, \( p_i = \{e_1; e_2; \ldots; e_n\} \subseteq P \) of length i, where \( P \) is a sequential pattern of the form \( \{e_1; e_2; \ldots; e_n\} [1 \leq i \leq n] \), the Occurrence Vector of any of the event \( e_j [1 \leq j \leq i] \) within \( p_i \) and \( P \), is of the form \( v_j = \{x_1; x_2; \ldots; x_t\} \), where \( t \) is the number of occurrence of \( e_j \) in \( P \) with \( x_j [1 \leq j \leq t, 1 < c < n] \) is the EId/Starting position/index of \( e_j \) in \( P \). For convenience, \( EId \) means the event identifier, which represents the position of the event in a string. For all possible valid sequential sub-pattern instance of \( p_i \) that is, \( p_i \subseteq P \), the Event_Id sequence will be of the form \( EId_{p_1}; EId_{p_2}; \ldots; EId_{p_l} \). Here \( EId_{p_1} [1 \leq i \leq l] \) represents the Event_Id/starting position of the ith event \( e_i \) in \( P \). Hence, \( EId_{p_l} \) will be the occurrence position of first event of \( P \). Conceptually for all instance of \( p_i \) that is, \( p_i \subseteq P \), the \( EId_{p_l} \) is the starting position of the pattern \( p_i \) and collectively all instances of \( EId_{p_l} \) form a set of occurrence position of \( p_i \) within \( P \). This set is known as Occurrence Vector of \( p_i \) which is by Definition 5, the Occurrence Vector of \( e_i \) in \( P \), that is, first event. Hence, it is proved that the Occurrence Vector of any pattern is same as its first event.

The occurrence vector of the mined patterns is the occurrence vector of the first character of the mined patterns. Suppose for string “abcdabcd” we can mine pattern \( \{ab\} \). The occurrence vector of the pattern \( \{ab\} \) is [0,4] which is the occurrence vector of the first event \( \{a\} \), that is, [0,4]. Now, the following Lemma 2 deals with the fact that the difference vector can indicate the allowable number of star events in between two specific important events, which accelerates the performance of our proposed algorithm.

**Lemma 2.** For a pattern \( p = \{x_1; x_2; \ldots; x_n\} \subseteq P \), if any of the Difference Vector\(=N \), then \( M = N – 1 \) number of \( \{*\} \) events \( \{*\} \) can be added in between the sub-patterns \( \{x_{1}; x_{2}; \ldots; x_{n}\} \) and \( \{x_i\} \), where \( P \) is a set of patterns with length i.

**Proof.** According to Lemma 1, Occurrence Vector of a pattern like \( p = \{x_1; x_2; \ldots; x_n\} \) represents the position of \( x_i \) in the actual event sequence and the Difference Vector for the pattern \( p \) is the difference between the Occurrence Vector of the two sub-patterns \( p_1 = \{x_1; x_2; \ldots; x_{i-1}\} \) and \( p_2 = \{x_i\} \) as described in Definition 9. In other words, say the difference between the Occurrence Vector, \( \text{Occurrence Vector}_{p_1} = m \) and Occurrence Vector, \( \text{Occurrence Vector}_{p_2} = n \) within \( p \). Therefore, there are certainly \( (n – m – 1) \) number of events within \( p_1 \) and \( p_2 \). When \( n > m + 1 \), that is, \( p_1 \) and \( p_2 \) are consecutively co-located, then no \( \{*\} \) events can be added. In the same way, when \( n > m + 1 \), non-zero number of \( \{*\} \) events can be added. And \( (n – m – 1) \) is numerically equivalent to \( N – 1 \), where \( N \) is the Difference Vector of \( p \).

As for example, we can apply maximum event skipping threshold value to skip the number of events in forming various patterns from \( T \). From occurrence of \( \{ax\} \) we find that the occurrence vectors of \( \{ax\} \) are [0,3], [4,6], [8,9] and [13,15]. So differences are 3, 2, 1 and 2 respectively. Hence, probable search patterns can be \( \{ax\} \) and \( \{ax\} \) with the occurrence vector is [0,4, 8, 13].

Up to this point, we have found the mining patterns of a specific length, we will now calculate the occurrence vector of those patterns.

3.2.2.2. Generating the occurrence vectors. After converting the whole time series database into string \( S = \{e_1; e_2; \ldots; e_n\} \) where the string contains the ascii characters like \( \{A,B,\ldots;Z,a,b,\ldots;Z\} \), the string is searched for the occurrence vector of each of the unique symbols that represents a specific event. For each of the event in the generated string, we record the event identifier, which represents the position of the event in a string and serves as the
time stamp within the string, and store them as Occurrence Vector for single length pattern. The event identifier is the position of the event in the generated string and serves as the time stamp within the string. While joining two patterns, the Event_Id of the ith event in the string is generated.

To find the occurrence vector of length two and the event identifiers of length two patterns in next pass, the proposed algorithm joins the all pairs of single events and their corresponding event identifiers. As in Lemma 1, the Occurrence Vector of length-2 patterns are the Occurrence Vector of the first event within that pattern. Similarly we can grow the pattern from length two to length three by joining the two length sub-patterns using the joining process described earlier and the Occurrence Vector is same as the first event of any particular event. That is, we can construct length-i patterns by joining two length-(i-1) patterns by maintaining the joining process and generate the Occurrence Vector of any pattern as stated in Lemma 1.

To clearly get the underlying idea, consider the time series database shown in Table 1, for which the string “accx acxd axdd bacx” can be generated by applying Discretization technique. Then, for the unique events within the string, i.e., a, b, c, d and x, Occurrence Vectors can be generated using the process described above. The intended reader can find this in first column of the Table 2. The table has three columns with heading “Patterns of Length1”, “Patterns of Length2” and “Patterns of Length3”. Each column is containing patterns of length specified in their heading, along with their Occurrence Vector, Difference Vector and EID i.e. Event Identifier for each event within that particular pattern.

From the generated patterns, we can now mine the different types of patterns with don’t care events. Using Table 3, we can mine patterns and their corresponding occurrence vectors. The Table 4 is showing only the mined patterns with the corresponding occurrence vectors with size more than 1. After finding occurrence vectors of all the mining patterns, we can apply any of the periodic pattern mining algorithm which will report the period. The proposed algorithm apply the periodicity detection algorithm which has been used in the suffix tree based algorithm. The overall pseudo code of the proposed algorithm is shown in Fig. 4.

For illustration, we have shown in Table 5, the generated patterns, their corresponding Event Identifiers, Occurrence vectors, difference vectors and mined patterns (if possible to generate) for the string “accx acxd axdd bacx” found by discretizing the database of Table 1 to search for patterns of the form a(x)^h, where 0 ≤ h ≤ 2, that is, “a*x”.

### 3.3. Algorithm description and analysis

Fig. 4 contains the pseudo code of our proposed algorithm. In line number 2, the proposed algorithm scans the entire database.

<table>
<thead>
<tr>
<th>EID (a)</th>
<th>EID (b)</th>
<th>EID (c)</th>
<th>EID (d)</th>
</tr>
</thead>
<tbody>
<tr>
<td>[0.4], [0.1], [0.2], [0.3], [0.5], [0.6], [0.7]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[4.5], [4.6], [4.7]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[1.4], [1.5], [1.2], [1.3], [1.6], [1.7], [5.6], [5.7]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[2.4], [2.5], [2.6], [2.3], [2.7], [6.7]</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[3.4], [3.5], [3.6], [3.7]</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Mined pattern</th>
<th>Occurrence vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>a-c</td>
<td>[0, 4]</td>
</tr>
<tr>
<td>ab</td>
<td>[0, 4]</td>
</tr>
<tr>
<td>bc</td>
<td>[1, 5]</td>
</tr>
<tr>
<td>a+b+d</td>
<td>[0, 4]</td>
</tr>
<tr>
<td>cd</td>
<td>[2, 6]</td>
</tr>
<tr>
<td>b+d</td>
<td>[1, 5]</td>
</tr>
<tr>
<td>a+c+d</td>
<td>[0, 4]</td>
</tr>
<tr>
<td>hcd</td>
<td>[1, 5]</td>
</tr>
<tr>
<td>ab-d</td>
<td>[0, 4]</td>
</tr>
<tr>
<td>abc</td>
<td>[0, 4]</td>
</tr>
<tr>
<td>abcd</td>
<td>[0, 4]</td>
</tr>
</tbody>
</table>

Then performs the discretization operation to construct the string S. Thus, it converts the whole time series database into string S. Here, within the string, each of the ASCII characters in the set [a-z][A-Z] represents each of the independent event. In the "for" loop of line numbers 3 to 5, after processing the generated string, the algorithm constructs the occurrence vector for each of the independent event.

As an example, consider a time series database is converted to a string T = {abcde abce abced} with period value 5. The proposed algorithm will extract the occurrence vector of all the independent events. We will find the following occurrence vectors for each of the event.

\[
a = (0, 5, 10, 13) \\
b = (1, 6, 11, 15) \\
c = (2, 7, 8, 12, 16, 17) \\
d = (3, 18) \\
e = (4, 9, 14, 19)
\]

The following Lemma 3, states the most robust feature of our proposed algorithm, that is, we can prune lots of search spaces in level by level approach, required to generate exclusive patterns.

**Lemma 3.** To determine the periodicity of a user provided searching pattern, \( \Phi = \{x_0(x^h), x_1(x^h), \ldots, x_{n-1}(x^h)\} \) with the length of the pattern, \( L = n + \sum_{i=0}^{n-1} |A_i| \), the pattern that needs to be generated is \( \Phi' = \{x_0,x_1,\ldots,x_{n-1}\} \) and the algorithm needs to run up to maximum \( n \) number of phases.

**Proof.** According to problem definition, we can add up to \( \theta \) number of intermediate don’t care events in between any two important events within a generated pattern and according to Lemma 2 the number of intermediate don’t care events can be from 0 up to minimum of \( \theta \) and difference vector value of the two important events in the generated string. Hence, in our proposed algorithm, we construct length \( j \) patterns with only important events in every \( i \)th phase and can generate our expected searching patterns by adding necessary and allowable number of don’t care events using the difference vector. As a result, we can get all of the patterns including the patterns with star events without considering the joining and periodicity mining of don’t care events oriented patterns. Hence, for a pattern \( \Phi \), we need to compute only those patterns without considering star events and the tight bound on the length of must to be mined patterns and number of phases will exactly \( n \). □

In at line 7, a new string \( \Phi' \) is calculated from \( \Phi \) by eliminating stars using the function \( ELIMINATE_\text{STARS}(\ ) \) and in line number 8 the required number of phase for the algorithm is calculated using the function \( LENGTH(\ ) \), which is the length of

<table>
<thead>
<tr>
<th>Mined Pattern</th>
<th>Occurrence Vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>a-c</td>
<td>[0, 4]</td>
</tr>
<tr>
<td>ab</td>
<td>[0, 4]</td>
</tr>
<tr>
<td>bc</td>
<td>[1, 5]</td>
</tr>
<tr>
<td>a+b+d</td>
<td>[0, 4]</td>
</tr>
<tr>
<td>cd</td>
<td>[2, 6]</td>
</tr>
<tr>
<td>b+d</td>
<td>[1, 5]</td>
</tr>
<tr>
<td>a+c+d</td>
<td>[0, 4]</td>
</tr>
<tr>
<td>hcd</td>
<td>[1, 5]</td>
</tr>
<tr>
<td>ab-d</td>
<td>[0, 4]</td>
</tr>
<tr>
<td>abc</td>
<td>[0, 4]</td>
</tr>
<tr>
<td>abcd</td>
<td>[0, 4]</td>
</tr>
</tbody>
</table>
the new string $\Phi$. From line number 9 the proposed algorithm starts to execute multiple passes using the "while" loop of line numbers from 10 to 48. The tremendous flexibility of the proposed algorithm permits the user to search for the patterns like \{abc $\backslash e$\}. Though the proposed algorithm needs to check for periodicity for the pattern with length 5 but there is one don't care event. Hence, only 4 level of pattern generation and searching is required. Hence, if the search pattern is $\Phi = \text{"abc}e\text{"}$ then $\Phi = \text{"abce"}$ which is of length 4.

In each pass $i$, line number 11 will generate patterns of length $i$ and its corresponding occurrence vector by joining any two lower length $i - 1$ patterns using the condition, that is, $i - 2$ length suffix of the first pattern should be equal to the $i - 2$ length prefix of the second pattern. In line number 11 the suffix and prefix is calculated using the functions $\text{SUFFIX}(\cdot)$ and $\text{PREFIX}(\cdot)$ respectively, where both take the pattern and length of the prefix/suffix as parameter. The proposed algorithm will execute until the user defined number of pass or it will execute until the pass where the generated

---

**Fig. 4.** The pseudocode of the proposed algorithm.
occurrence vector of the generated string within this pass is null. In each of the pass the algorithm will generate patterns of length \( i \) by joining \( i - 1 \) length patterns where pattern length \( i > 2 \), so that the event identifiers of the \( i - 1 \) length patterns maintain the sequential ordering.

The “for” loop described from line number 13 to 23, For each generated patterns \( \{e_1, e_2, \ldots, e_i\} \), \( i \geq 2 \) calculate occurrence vector of that pattern by joining the occurrence vectors of the lower length \( i - 1 \) patterns, that is, by joining the generated patterns \( \{e_1, e_2, \ldots, e_{i-1}\} \), where \( \text{Event Id}(X_{i-1}) < \text{Event Id}(X_i) \). Here, the pattern \( X_{i-1} \) is lower length patterns than \( X_i \). Then, for each occurrence vector, difference vector \( Z_1, Z_2, Z_3, \ldots, Z_1 \) is calculated, where \( Z_1 = X_i - X_{i-1} \). We take the difference between any two successive occurrence vector elements leading to another vector called the difference vector. The “for” loop described from lines 17 to 22 performs huge optimization in the search space by terminating the periodicity searching operation in case of a difference vector value among any two important event found greater than the number of star events between those two events within the search pattern. As an example, say we need to search for “ababab” but the difference vector value is 3 or greater. Then, we can conclude that the search pattern is absent within the time series database and no need to search the pattern for periodicity.

Hence, the algorithm will generate 2 length patterns and join the occurrence vectors of each of the independent event to generate the occurrence vector of length 2 pattern.

\[
\{ab\} = \{0,1\}, \{5,6\}, \{10,11\}
\]

We will not join 0 with 6, due to the period value is 5. Moreover, the event \( b \), which occurs in position 6 means it occurs in second period and \( a \) which occurs in position 0 means it occurs in first period. That implies, we will only join the event identifiers of those events which are belonging to the same period. In this way, the following patterns and event identifiers are generated.

\[
\begin{align*}
\{ac\} & = \{0,2\}, \{5,7\}, \{10,12\} \\
\{ae\} & = \{0,4\}, \{5,9\}, \{10,14\} \\
\{bc\} & = \{1,2\}, \{6,7\}, \{6,8\}, \{11,12\}, \{15,16\}, \{15,17\} \\
\{be\} & = \{1,4\}, \{6,9\}, \{11,14\}, \{15,19\} \\
\{ce\} & = \{2,4\}, \{7,9\}, \{8,9\}, \{12,14\}, \{16,19\}, \{17,19\}
\end{align*}
\]

In the “for” loop described from line numbers 25 to 38, the proposed algorithm performs the periodicity detection approach to find out the periodicity of the mined patterns. For each of the mined patterns, it reports all possible periods and also counts the number of times the particular period is reported. For each occurrence vector of size \( i \) for pattern \( p \), the proposed algorithm applies the periodicity detection approach. It applies the logic described in Algorithm 1 in Rasheed et al. (2011). In particular, any other approach can also be applied to detect the periodicity of the mined patterns.

In line numbers 39 to 45, using the “for” loop, the proposed algorithm approaches for the next passes by filtering the generated patterns based on user provided periodicity confidence threshold. Here, the function \( \text{CONFIDENCE}(p) \) is used to calculate the confidence of the generated patterns within the time series database to filter patterns based on user preferences. The calculation, essential for calculating confidence is described in Definition 6 of Section 2.1.

We have set up our minimum confidence threshold as 75%. That means, if any pattern has actual periodicity 3 or greater out of total 4 within the discretized event sequence string with period value \( p = 5 \) then we will consider this pattern as frequent periodic pattern, since it satisfies the minimum confidence threshold. All of the above patterns generated in phase 2 gratify minimum confidence 75%. Hence, we will forward to the phase 3, where following pattern of length 3 will be generated.

\[
\begin{align*}
\{abc\} & = \{0,1,2\}, \{5,6,7\}, \{5,6,8\}, \{10,11,12\} \\
\{abe\} & = \{0,1,4\}, \{5,6,9\}, \{10,11,14\} \\
\{ace\} & = \{0,2,4\}, \{5,7,9\}, \{10,12,14\} \\
\{bce\} & = \{1,2,4\}, \{6,7,9\}, \{6,8,9\}, \{11,12,14\}, \{15,16,19\}, \{15,17,19\}
\end{align*}
\]

The above patterns generated in phase 3 convince minimum confidence 75%. Hence, we will proceed to the phase 4 and generate following pattern of length 4.

\[
\begin{align*}
\{abc\} & = \{0,1,2,4\}, \{5,6,7,9\}, \{5,6,8,9\}, \{10,11,12,14\}
\end{align*}
\]

We have generated minimum required patterns along with their occurrence and difference vector. Lemma 4, stated below, describes a way of inserting don’t care events and all possible patterns that can be generated from the already mined patterns.

**Lemma 4.** Let, \( P \) is a set of patterns and \( p = \{e_1, e_2, \ldots, e_n\} \in P \) is a pattern with the maximum event skipping threshold is \( \theta \), then the patterns that can be mined using the proposed algorithm, are all possible sub-sequences of \( p \), of the form \( \{e_1(s)^0, e_2(s)^0, \ldots, e_n(s)^0, e_i\} \subseteq p \), \( a_k \in [0, \theta] \) and \( 1 \leq k \leq n - 1 \leq i - 1 \)

**Proof.** From Lemma 2, we found that, for a pattern, \( p = \{e_1, e_2, \ldots, e_n\} \), for any value \( n \) of the Difference Vector \( r_{\nu} \) \( (n - 1) \) number of \((s)\) events can be added within the two sub-patterns \( p_1 = \{e_1, e_2, \ldots, e_{i-1}\} \) and \( p_2 = \{e_i\} \) of \( p \). Moreover, from the problem definition, we know that, we will consider the number of \((s)\) events within the range \([0, \theta]\). However, it can be observed that, the maximum number of \((s)\) intermediate events in between first and last event
can be \((i - 2)\) for \(p\). Therefore, obviously \(\theta \leq (i - 2)\) is true. Now, consider any pattern \(p_i = [e_1, e_2, \ldots, e_i] \subseteq p\) with \(\text{Difference Vector}_i = \{A_1, A_2, \ldots, A_j\}\). Hence, for \(A_i, [1 \leq i \leq j]\), \(\min(A_i - 1, \theta)\) number of events can be added into the sub-pattern, \(p_i = \{e_1, e_2, \ldots, e_i\}\) and \(p_j = \{e_i\}\) of \(p\). This is same for any pattern, \(p_i \subseteq p\). Therefore, we can conclude that Lemma 2 shows the all possible patterns that will be generated by the proposed algorithm.

After termination of the while loop of lines 10 to 48, the proposed algorithm generates patterns of user interest in lines 49 to 54 by adding don’t care events within important events based on the difference vector of those two events in the pattern, which is the special feature of our proposed algorithm. Moreover, consider a case where a manager needs to search his marketing transaction database, where within a same period, more than one instances of an event (as instance, transaction by check) can occur and we can easily ignore the subsequent occurrences after taking the first one to perform huge optimization. Here, the line number 51 of the algorithm can be modified to perform such operation.

From phase 4, we will mine pattern \(\{abc\}\). Since, \(\{abc\}\) means the pattern with 0 or 1 intermediate don’t care event (as specified in problem definition of the proposed algorithm) then the occurrence vector of \(\{abc\}\) contains the occurrence vector of both \(\{abc\}\) and \(\{abc\}\). From the occurrence vector of \(\{abc\}\), we can determine the confidence level of both \(\{abc\}\) and \(\{abc\}\). However, by utilizing the difference vector, we can mine \(\{abc\}\) and its occurrence vectors separately.

As a consequence, the difference vector of \(\{abc\}\) for events ‘c’ and ‘e’ is \([2, 2, 1, 2]\). Hence, \(\{abc\}\) is present once in the whole string and exactly \(\{abc\}\) occurs three times. Now, suppose any user wants to compute the periodicity of same string but with different period value, then a minor change in computation sequence is enough, which can be easily and autonomously adopted by our proposed algorithm.

Moreover, if any user wants to search for a different string like \(\{abc\}\), \(\{abc\}\), \(\{abc\}\) or \(\{abc\}\) with various period value, then our algorithm still can compute periodicity only in 4 phases, due to the number of important events within search string is only 4, according to Lemma 3.

From the analytical point of view, we have seen the performance of the proposed algorithm is controlled by three important and major parameters. These are:

1. Phases of the algorithm
2. Threshold value of the algorithm
3. Number of the unique symbols of the algorithm

Speed of the proposed algorithm depends on the phase within which it will be executed, if the threshold value of the algorithm and the number of the unique symbols of the algorithm is fixed. The speed of the proposed algorithm depends on the threshold value, if phase and the number of the unique symbols of the algorithm is fixed. Moreover, the speed of the proposed algorithm depends on the number of the unique symbols, if phase and the threshold value of the algorithm is fixed.

4. Experimental results

The proposed algorithm is directly applied on the real life time series datasets. Before applying the proposed algorithm and existing algorithm on the time series database, the database is modified to control few parameters such as the length of the time series databases, the number of unique symbols and the time instances. Based on the criteria of time series data in time series database, the discretization range and threshold value are changed. The proposed algorithm is compared with the existing suffix tree based algorithm in a particular area, that is, in generating different kinds of patterns. The comparison showed that suffix tree cannot skip the intermediate events and omit various interesting patterns which were generated with the proposed algorithm.

4.1. Experimental environment settings

The suffix tree based existing algorithm and proposed algorithm are implemented in Java programming language using eclipse Helios IDE and tested extensively in Windows XP operating system. The experimental results are shown using a 2 GB RAM and core 2 duo processor. As datasets are concerned, we have used datasets which contains diabetes patient records, which were obtained from two sources: an automatic electronic recording device and paper records. This is found from the data source UCI machine learning repository (Michael Kahn, 2010).

4.2. Performance analysis

In this section, we have discussed the results found after applying the proposed algorithm in the time series database that contains diabetes data. The time series database, which is collected from different sources, was modified based on different parameters. During modification, addition and deletion from the original time series database, the parameters which were controlled are: alphabet size, data size and period size that means the number of unique symbols in the data, the length of the time series database which is the number of symbols of the data and the period size respectively.

During the time estimation the parameters listed above were changed and effect of the time estimation is examined. Here the maximum event skipping threshold, \(\theta\) plays very important role. Actually, the threshold controls the size of the patterns which is generated to be checked for the periodicity.

4.2.1. Time estimation without considering the threshold, \(\theta\)

Table 6 was found, when the number of events (unique symbol) within the time series database is 4 without threshold. The first column contains the length of various time series database (in number) and second column contains the time of the execution to generate the occurrence vectors of the different patterns (in seconds). Hence, we can observe that time increases sharply when the number of data in datasets that means the length of the time series database is 50. Hence, when we will not apply any threshold, the time is increased in proportion to the length of the time series database.

<table>
<thead>
<tr>
<th>Table 6</th>
<th>The table of time estimation when the number of events (unique symbol) within the time series database is 4 without threshold.</th>
</tr>
</thead>
<tbody>
<tr>
<td>The length of the time series database (in number)</td>
<td>The time of the execution to generate the occurrence vector (in seconds)</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>20</td>
<td>4</td>
</tr>
<tr>
<td>32</td>
<td>17</td>
</tr>
<tr>
<td>45</td>
<td>36</td>
</tr>
<tr>
<td>50</td>
<td>106</td>
</tr>
<tr>
<td>55</td>
<td>188</td>
</tr>
<tr>
<td>60</td>
<td>477</td>
</tr>
<tr>
<td>80</td>
<td>680</td>
</tr>
</tbody>
</table>
4.2.2. Time estimation considering the threshold, $\theta$

Next we examined the estimated time required to generate the occurrence vectors applying the threshold value. That means, a certain number of the intermediate events can be skipped. Table 7 is showing the time estimation when the number of events (unique symbols) within the time series database is varied and threshold value is varied. It implies, the number of intermediate events that can be skipped is varied.

In first column of Table 7, the first sub-column holds the length of the time series database (in number) and second sub-column holds the time of the execution to generate the occurrence vectors of the different patterns (in second). Here, we can observe that the time required to generate the occurrence vectors depends on the threshold values applied. The estimated time is lower in case of the threshold value not applied.

Table 7
The table of time estimation in the varying number of events (unique symbol) within which the time series database and in varying threshold values.

<table>
<thead>
<tr>
<th>The number of events (unique symbol) within the time series database is 4 and threshold value is 3</th>
<th>The number of events (unique symbol) within the time series database is 8 and threshold value is 3</th>
<th>The number of events (unique symbol) within the time series database is 8 and threshold value is 6</th>
</tr>
</thead>
<tbody>
<tr>
<td>The length of the time series database (in number)</td>
<td>The length of the time series database (in number)</td>
<td>The length of the time series database (in number)</td>
</tr>
<tr>
<td>The time of execution to generate the occurrence vector (in seconds)</td>
<td>The time of execution to generate the occurrence vector (in seconds)</td>
<td>The time of execution to generate the occurrence vector (in seconds)</td>
</tr>
<tr>
<td>96</td>
<td>1</td>
<td>64</td>
</tr>
<tr>
<td>100</td>
<td>2</td>
<td>72</td>
</tr>
<tr>
<td>120</td>
<td>5</td>
<td>80</td>
</tr>
<tr>
<td>130</td>
<td>6</td>
<td>88</td>
</tr>
<tr>
<td>140</td>
<td>9</td>
<td>96</td>
</tr>
<tr>
<td>150</td>
<td>12</td>
<td>104</td>
</tr>
<tr>
<td>160</td>
<td>15</td>
<td>120</td>
</tr>
<tr>
<td>170</td>
<td>20</td>
<td>130</td>
</tr>
<tr>
<td>180</td>
<td>25</td>
<td>140</td>
</tr>
<tr>
<td>190</td>
<td>31</td>
<td>150</td>
</tr>
<tr>
<td>200</td>
<td>46</td>
<td>160</td>
</tr>
<tr>
<td></td>
<td></td>
<td>170</td>
</tr>
<tr>
<td></td>
<td></td>
<td>180</td>
</tr>
<tr>
<td></td>
<td></td>
<td>190</td>
</tr>
<tr>
<td></td>
<td></td>
<td>200</td>
</tr>
</tbody>
</table>

Table 8
The missing patterns in suffix tree based algorithm for string “abcababbb”.

<table>
<thead>
<tr>
<th>Missing patterns in suffix tree</th>
<th>Occurrence vector of missing patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a + a$</td>
<td>[0.3]</td>
</tr>
<tr>
<td>$a + ab$</td>
<td>[0.3]</td>
</tr>
<tr>
<td>$a + bb$</td>
<td>[0.3]</td>
</tr>
<tr>
<td>$a + b$</td>
<td>[0.3]</td>
</tr>
<tr>
<td>$a + b$</td>
<td>[0.3]</td>
</tr>
<tr>
<td>$a + b$</td>
<td>[0.3]</td>
</tr>
<tr>
<td>$a + b$</td>
<td>[0.3]</td>
</tr>
<tr>
<td>$a + b$</td>
<td>[0.3]</td>
</tr>
<tr>
<td>$b + a$</td>
<td>[1.4]</td>
</tr>
<tr>
<td>$b + ab$</td>
<td>[1.4]</td>
</tr>
<tr>
<td>$b + ab$</td>
<td>[1.4]</td>
</tr>
<tr>
<td>$b + b$</td>
<td>[1.4]</td>
</tr>
<tr>
<td>$b + b$</td>
<td>[1.4]</td>
</tr>
</tbody>
</table>

4.3. Performance comparison with suffix tree based algorithm

We have generated patterns for the time series database after modifying slightly, containing diabetes data sets, using suffix tree based algorithm and proposed algorithm. Here the proposed algorithm is executed within pass 4. We have observed that the proposed algorithm can generate different patterns which is not possible to be generated by suffix tree. These were known as missing patterns. Because the occurrence vector of these patterns are different.

In Table 8, the column named missing patterns are showing all of the missing patterns which cannot be generated by suffix tree based algorithm. For example, the suffix tree can generate two patterns “abcababbb” with occurrence vector [0] and “abbabb” having occurrence vector [3]. However, the proposed algorithm generates a pattern “ababbb” having occurrence vector [0, 3]. Look at the occurrence vectors generated by suffix tree based algorithms where exists no pattern with occurrence vector [0, 3]. Hence, we have proved that the suffix tree based algorithm missed the patterns “ababbb”, because in the user’s point of view it is extra added flexibility that the user wants to check the periodicity of the pattern “ababbb”, where two intermediate events have been skipped. The reason is, user finds these two events as don’t care events or user is concerned only in generating 4 length patterns where first and last events are only a. The user is not concerned in the intermediate two events. We can claim that [0, 3] is missing occurrence vector, because suffix tree based algorithm fails to generate the occurrence vector [0, 3].

At this point, we have developed 2 criteria for reporting a pattern is missing in suffix tree.

1. A pattern is being generated by the proposed algorithm whose occurrence vector is not reported by the suffix tree for any of its generated pattern.

2. A pattern is being generated by the proposed algorithm whose occurrence vector has been reported by the suffix tree based algorithm but the suffix tree has reported this occurrence vector for the smaller length pattern and same occurrence vector has been reported by the proposed algorithm for higher length pattern than the pattern generated by suffix tree.

As an example, for pattern “abcababbb”, suffix tree has reported occurrence vectors for “$b^2$” = [1, 5]. However, the proposed algorithm can report “$b^2d^2$” = [1, 5]. Then “$b^2d^2$” is missing patterns by the suffix tree. Because, from lower length “$b$”, we cannot decide the next events which might be important for user.
4.4. Performance comparison based on the number of periods being reported

We will now compute the performance of our proposed periodicity detection algorithm with respect to number of reported periods. We have already mentioned that for calculating periodicity, we have applied the same periodicity detection algorithm used in the suffix tree based algorithm. The proposed algorithm can generate various interesting patterns which are missing in generated patterns by the suffix tree. Moreover, we have observed that the occurrence vectors of these patterns are different.

Tables 10 and 11 are showing the comparison in between the number of different periods being reported by the proposed algorithm and suffix tree based algorithm, for the patterns generated with length more than two, based on the different parameters such as the length of time series database and the total number of unique symbols. From Tables 10 and 11 we can observe that the proposed algorithm can report more periods than the existing suffix tree based algorithm. If we run the proposed algorithm more than three phases, it will report more periods.

As we have found in our motivating scenario in Section 1, if we search for any pattern like “a*xc” where the user is interested about only the first and third event and don’t care about the intermediate second event. Using our proposed algorithm, then “abxc”, “axcc” and “adxcc” can be reported but existing algorithms don’t provide such flexibilities to search for such pattern. Moreover, we have to provide specifically the pattern “abxc” or “axcc” or “adxcc”. And only the periodicity for that specified pattern will be checked. Hence, the ratio among the number of interesting reported patterns using existing algorithm and proposed algorithm is \( \frac{1}{2} \). More important fact is that the user interaction and required domain knowledge of the user is abated. Another important fact is that we can also mine some more interesting and exclusive patterns like “a{eq}X\}X”, “a{eq}X\}X” and so on which are excluded within the patterns generated by existing suffix tree based algorithm (Rasheed et al., 2011).

Moreover, from Tables 10 and 11 we can also test that the suffix tree based algorithm can miss various periods because it can not generate many periodic patterns. It is the most important contribution of the proposed algorithm. We can also observe that the proposed algorithm can cover all of the periods that can be reported by the suffix tree.

5. Discussion

As we have discussed in Section 1, our proposed algorithm can generate exclusive and interesting patterns, those are missing in generated patterns by existing algorithms. These exclusive patterns can be used to solve various real life problems. This section deals with the applications of our proposed algorithm and few scenarios where our algorithm proved to be useful.

Suppose a survey on road and transport of a city is to be conducted to find out the busiest road segments on a certain month. As a general query, it would be specified by specific time period. There could be some exceptional cases at some time interval during any festival. Generally at that time, there could be excessive traffic which is exceptional and can be neglected within that time period. After that festival, there could be a lower amount of traffic on the roads of the city. This case is also exceptional and need not to be considered. Hence, the query of the official could be something like a sub-sequence of time interval with don’t care events.

Again consider, a manager of a company needs to check the transactions for predicting next transactions that occur through cash and credit card. But the transaction could also contain some payment with check and with due payment. At this point, the query of the manager should neglect all the intermediate transactions which occur through check and with due payment. These negligible intermediate events can be represented as don’t care events in a query pattern for searching the database using the proposed algorithm. To perform such analysis on this type of database we can use our algorithm in a more efficient way.

This algorithm can also be applied in a weather-forecast center to predict the humidity for a country with the knowledge in mind.
that during winter the humidity needs not to be considered. Hence, if the time interval contains any month’s information which belongs to winter then that information can be neglected. Hence the database of month based temperature can be searched by our algorithm to predict the information by generating a pattern with the negligible period represented as don’t care event.

As another application of the proposed algorithm, suppose employer of a company wants to analyze the bandwidth used by the employees to browse the internet to perform assigned task within office hours. However, in the lunch break, tea break and in over-time period the information related to bandwidth used by the employees is neglected because this time is not covered within actual working hours. As our closing example, suppose the authority of the stock market wants to perform lower stock rate analysis for the products within specific time interval. However, within that time period, the stock market price for specific product may be higher than certain range. This stock market prices, which are higher than certain range are neglected within that time period. The negligible event can be considered as don’t care event as usual and the necessary information can be searched from the database by generating a pattern with such don’t care event.

Finally, we can conclude that our proposed algorithm can be applied in a large number of real-life applications where we can search for information with/without don’t care events.

6. Conclusions

To mine periodicity in time series database, various algorithms have been proposed. Most of the algorithms have specific limitations and flexibilities, such as some algorithms can detect only partial periodicity, some can detect symbol periodicity. In Rasheed et al. (2011), authors proposed a tree based algorithm which can mine periodicity within time series database efficiently. However, we have observed that the algorithm in Rasheed et al. (2011) failed to generate and check some interesting patterns which include user’s choice to disregard some intermediate events. As we have shown in Section 1 that how event skipping can be useful in generating more interesting user specified patterns. Therefor, in this paper we have presented an algorithm where the single algorithm can find symbol, sequence (partial periodic), and segment (full cycle) periodicity in the time series database. One of the major limitations for the existing periodic pattern mining algorithms is, the existing algorithms assume that periodicity rate is user specified. The proposed algorithm has overcome this limitation. Moreover, the existing algorithms face difficulty or wastes time to make decision whether there exists symbol, partial or full cycle periodicity. However, the proposed algorithm can report the symbol, partial or full cycle periodicity in one run. As well as, the proposed algorithm is more user friendly, interactive and efficient enough in mining more interesting patterns.

References


